

**UNIVERSITY OF MICHIGAN**  
**College of Engineering**  
**Curriculum Committee Meeting**  
**Tuesday, January 14 – Email Vote**  
**Email Vote**

**Attending:**

**Call to Order:**

**Adjourned:**

**AGENDA**

1. 11.19.19 Meeting Minutes Approval: **APPROVED**

**CARF SUMMARIES**

PAGE	SUBJECT	COURSE #	ACTION	SUMMARY	EFFECTIVE TERM	MIN. GRADE REQ. FOR ENF. PREPREQ	APPROVED	NOTES & REVISIONS	TABLED
5	ROB	502	NEW		FA 2020		X		
17	ROB	511	NEW		FA 2020		X		
104	MECHENG	395	MOD	change to enforced prerequisites	FA 2020	C-	X		
107	MECHENG	450	MOD	change to enforced prerequisites	FA 2020	C	X		

**UNIVERSITY OF MICHIGAN**  
**College of Engineering**  
**Curriculum Committee Meeting**  
**Tuesday, January 14 – Email Vote**  
**Email Vote**

**Attending:**

**Call to Order:**

**Adjourned:**

**AGENDA**

**CARF SUMMARIES**

<b>PAGE</b>	<b>SUBJECT</b>	<b>COURSE #</b>	<b>ACTION</b>	<b>SUMMARY</b>	<b>EFFECTIVE TERM</b>	<b>MIN. GRADE REQ. FOR ENF. PREPREQ</b>	<b>APPROVED</b>	<b>NOTES &amp; REVISIONS</b>	<b>TABLED</b>
2	ROB	502	NEW		FA 2020				
14	ROB	511	NEW		FA 2020				
101	MECHENG	450	MOD	change to enforced prerequisites	FA 2020	C-			
104	MECHENG	395	MOD	change to enforced prerequisites	FA 2020	C			



# Course Approval Request Form

Office of the Registrar, University of Michigan

1210 ISA Building  
500 S. State Street  
Ann Arbor, MI 48109-1382  
Phone: 734.763.2113  
Fax: 734.936.3148  
ro.curriculum@umich.edu  
ro.umich.edu

CHECK APPROPRIATE BOXES FOR ALL CHANGES

### Action Requested

- New Course
- Modification of Existing Course
- Deletion of Existing Course

Date of Submission: 2019-12-12  
Effective Term: Fall 2020

<input checked="" type="checkbox"/>	<b>Course Offered</b> <input checked="" type="checkbox"/> Indefinitely <input type="checkbox"/> One term only	<b>RO USE ONLY</b> Date Received: Date Completed: Completed By:
-------------------------------------	---	--

### CURRENT LISTING

### REQUESTED LISTING

<input checked="" type="checkbox"/>	Dept (Home):			Dept (Home): Robotics		
	Subject:			Subject: ROB		
	Catalog:			Catalog: 502		
	<input type="checkbox"/> Course is Cross-Listed with Other Departments					
<input type="checkbox"/>	Department	Subject	Catalog Number	Department	Subject	Catalog Number
<input checked="" type="checkbox"/>	Course Title (full title)			Course Title (full title) Programming for Robotics		
<input checked="" type="checkbox"/>	Abbreviated Title (20 char)			Abbreviated Title (20 char) Programming for ROB		
<input checked="" type="checkbox"/>	Course Description (Please limit to 50 words and attach separate sheet if necessary) Graduate level project-based C programming and computer science course for Robotics engineers. Topics include data representation, memory concepts, debugging, recursion, search, abstractions, threading, and message passing. The average student will have already written MATLAB programs about 250-500 lines long and will have basic familiarity with C syntax.					
<input checked="" type="checkbox"/>	Full Term Credit Hours			Half Term Credit Hours		
	Undergraduate Min:		Graduate Min: 3	Undergraduate Min:		Graduate Min:
	Undergraduate Max:		Graduate Max: 3	Undergraduate Max:		Graduate Max:
<input checked="" type="checkbox"/>	Course Credit Type Rackham Graduate Student					
<input type="checkbox"/>	Repeatability			Repeatability		
	<input type="checkbox"/> Course is Repeatable for Credit			<input type="checkbox"/> Course is Y graded		
	Maximum number of repeatable credits:			<input type="checkbox"/> Can be taken more than once in the same term		

Subject: Robotics      Catalog: 502	
<input checked="" type="checkbox"/>	<b>Grading Basis</b> <input checked="" type="checkbox"/> Graded (A – E) <input type="checkbox"/> Credit/No Credit <input type="checkbox"/> Satisfactory/Unsatisfactory <input type="checkbox"/> Pass/Fail <input type="checkbox"/> Business Administration <b>Grading</b> <input type="checkbox"/> Not for Credit <input type="checkbox"/> Not for Degree Credit <input type="checkbox"/> Degree Credit Only
	<b>Add Consent</b> <input type="checkbox"/> Department Consent <input checked="" type="checkbox"/> Instructor Consent <input type="checkbox"/> No Consent
	<b>Drop Consent</b> <input type="checkbox"/> Department Consent <input checked="" type="checkbox"/> Instructor Consent <input type="checkbox"/> No Consent

	CURRENT LISTING	REQUESTED LISTING
<input type="checkbox"/>	Advisory Prerequisite (254 char)	Advisory Prerequisite (254 char)
<input type="checkbox"/>	Enforced Prerequisite (254 char)	Enforced Prerequisite (254 char)
<input type="checkbox"/>	Minimum grade requirement:	Minimum grade requirement:
<input type="checkbox"/>	Credit Exclusions	Credit Exclusions
<input checked="" type="checkbox"/>	<b>Course Components</b> <input type="checkbox"/> Lecture <input type="checkbox"/> Seminar <input checked="" type="checkbox"/> Recitation <input checked="" type="checkbox"/> Lab <input type="checkbox"/> Discussion <input type="checkbox"/> Independent Study	<b>Graded Component</b> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
		<b>Terms Typically Offered</b> <input checked="" type="checkbox"/> Fall <input type="checkbox"/> Winter <input type="checkbox"/> Spring <input type="checkbox"/> Summer <input type="checkbox"/> Spring/Summer

Cognizant Faculty Member Name: Ella Atkins & Ed Olson      Cognizant Faculty Member Title: Professor

**SIGNATURES ARE REQUIRED FROM ALL DEPARTMENTS INVOLVED (Please Print AND Sign Name)**

Contact Person: Denise Edmund      Email: dledmund@umich.edu      Phone: 7-2970

Curriculum Committee Member:	Print:	Date:
Curriculum Committee Chair:	Print:	Date:
Home Department Chair:	Print: Jessy Grizzle	Date: 12/13/19
Cross-Listed Department Chair:	Print:	Date:
Cross-Listed Department Chair:	Print:	Date:
Cross-Listed Department Chair:	Print:	Date:

**DEPARTMENTAL/COLLEGE USE ONLY**

**Current:****Requested:**Course Description

Graduate level project-based C programming and computer science course for Robotics engineers. Topics include data representation, memory concepts, debugging, recursion, search, abstractions, threading, and message passing. The average student will have already written MATLAB programs about 250-500 lines long and will have basic familiarity with C syntax.

Course Description

Graduate level project-based C programming and computer science course for Robotics engineers. Topics include data representation, memory concepts, debugging, recursion, search, abstractions, threading, and message passing. The average student will have already written MATLAB programs about 250-500 lines long and will have basic familiarity with C syntax.

Class Length

Full term

Class Length

Full term

Contact hours (lecture):Contact hours (lecture):Contact hours (recitation)Contact hours (recitation)Contact hours (lab)

6

Contact hours (lab)

6

**Additional Info:**Submitted by:

Home dept

Describe how this course fits with the degree requirements:

Core Course

ABET departmental program outcomes for undergraduate courses:

Not ABET accredited

Special resources of facilities required for this course:

computer lab

Supporting statement:

This course will serve as a core course for students entering the Robotics Masters/PhD program without a strong background in computer science and programming. This course will expose students to fundamental computer science topics through a series of complex projects that build up a robotic simulation environment from scratch. Students will learn to design, organize, and debug complex programs integrating various components they build over the course of the semester. This hands-on experience will prepare students to write and debug real robotic systems.

We anticipate students with diverse engineering backgrounds and familiarity with programming. The fundamental and intensive nature of this course will allow students to build a common base of computer science understanding that will support their future coursework, especially the programming components of the Robotics Systems Laboratory, ROB 550.

# ROB 599: Programming for Robotics | Fall 2019 | Michigan Robotics

Instructor: Acshi Haggemiller (acshikh), PhD Candidate  
Tu/We 1:30-4:30pm  
1620 Bob and Betty Beyster Building (CAEN lab in the hallway to DOW)

This whole site is a living document and subject to change.

## Introduction

This class is designed for engineering students who have a basic understanding of programming but haven't majored in computer science or taken a dedicated sequence of programming courses. **The goal of this class** is for students to learn how to 1) write programs from scratch that meet robotic system requirements; 2) organize programs into logical sections; 3) critique program design and implementation choices; 4) use appropriate debugging tools and methodology to efficiently understand and correct program behavior; and 5) use the command line to work with git and other relevant utilities and scripts.

As it is titled *Programming for Robotics*, we have tried to design the in-class problems and homework assignments to be relevant to common robotics situations and algorithms, with the greater goal of demystifying programming and avoiding black-box magic. To be relevant and exciting, we designed the homework assignments around building a robotics simulation environment. While there are many excellent libraries and tools available for this (ROS among them), we will figure it out for ourselves! The best way to learn programming is by programming, so there will not be any quizzes or exams, and algorithms and necessary math will be provided so you can focus on implementation and not derivation.

The class uses the C programming language. C is a relatively simple language that will help us understand the fundamentals of how computer programs work, without the language letting us take complicated features for granted. Although most robotics programming is done in languages like Python and C++, the fundamentals you learn in C will help you to better understand what is happening in those more complicated languages.

Most class sessions will follow a "labtute" format. We will start with a series of segments containing a short lecture, an instructor demo, individual work on a self-contained problem, and finally class solution evaluation. Students are encouraged to ask each other and the instructor for help, but only students

who have finished a problem should be looking at other students' code. In the evaluation, we will review several anonymous solutions and talk about their relative strengths.

Some class sessions will not have any formally scheduled instruction or problems. Instead, topics will be addressed on an as-needed basis, with the remaining time open for working on the homework assignments with instructor help. The homework assignments are intended to require about 4 hours per class session. In general, they will be due 1 week after the end of the topic section they were assigned in. For example, the first homework will be due before class session 5.

## Class Schedule

### Classes 1-3: Data representation

- Goals: 1) Inspect abstract data (e.g. pictures, text, plans) at the byte and bit level, and understand how changing low-level numbers affects high-level meaning. 2) Use the command line with git and the class submission system to get feedback.
- [Class 1: Using Linux and bash](#)
- [Class 2: Using git to commit and submit code; expressing logic](#)
- [Class 3: Arrays, ASCII, bytes, and GDB](#)
- [Homework 1: Polygonal collision detection, cryptogram](#)
- There are variety of C concepts that will not be explicitly covered in class! We are providing a [tutorial document](#) to help explain the necessary syntax and basic ideas so we can delve right into the good stuff!
- For an even gentler introduction to C, I highly recommend Harvard's CS50 lectures. Although the whole lectures can be long, they have good tables of contents on each lecture on YouTube, and work well at 2X playing speed. [This clip](#) focuses on compiling C, on using make, and on common compiler errors. [This one](#) is on the compilation process. If you want to follow along with their examples, you will need to use their [sandbox](#).

### Classes 4-7: Memory concepts and debugging

- Goals: 1) Determine when dynamic memory is appropriate and how to prevent and detect memory leaks. 2) Determine when pointers are necessary and reason about when they are valid. 3) Use feedback from GDB, Valgrind, and AddressSanitizer to fix memory and other bugs.
- [Class 4: Addresses, pointers](#)
- [Class 5: Malloc/free, debugging errors, and dynamic arrays](#)
- [Class 6: Linked lists](#)
- **Class 7:** As needed

- [Homework 2: Rasterizing bitmaps, Braitenberg vehicles](#)
- [This clip](#) talks about how data is stored in memory. [This one](#) talks about pointers. [This one](#) talks about malloc and free. [This one](#) talks about memory addresses and hexadecimal. [This one](#) is on stack overflows.

## Classes 8-10: Recursion and Search

- Goals: 1) Reason about and write recursive algorithms. 2) Use search algorithms with forward simulation to choose robot actions.
- [Class 8: Bisection search, midpoint method, recursion vs iteration](#)
- [Class 9: Tree search](#)
- **Class 10:** As needed
- [This clip](#) gives an overview of recursion and how the computer's stack is used to hold multiple versions of the same function in memory.
- [Homework 3: Equation parsing, robot chase](#)

## Classes 11-13: Object abstractions

- Goals: 1) Analyze algorithmic complexity and determine when it matters. 2) Choose data structures based on algorithm needs. 3) Separate and hide implementation from specification.
- [Class 11: Complexity/Big-O Notation](#)
- [Class 12: Hash tables](#)
- **Class 13:** As needed
- [Homework 4: Bigrams](#)

## Classes 14-17: Threading

- Goals: 1) Understand when threading is necessary and how to avoid using it unnecessarily. 2) Determine when variables may be subject to race conditions and how to prevent them. 3) Use threading for terminal input control.
- [Class 14: Basic threading](#)
- [Class 15: Race conditions, deadlock, mutexes](#)
- [Class 16: Terminal settings, I/O threading, manual robot control](#)
- **Class 17:** As needed
- [Homework 5: Live-tuning potential fields](#)

## Classes 18-20: Message passing and networking

- Goals: 1) Divide robotic systems into independent parts. 2) Coordinate program communication across network nodes. 3) Use logging and playback features to debug specific modules.
- [Class 18: LCM/ROS basics, hybrid architectures](#)
- [Class 19: Networking](#)
- **Class 20:** As needed



- [Homework 6: Split project into communicating processes](#)

## **Classes 21-23: Special topics**

- **Class 21:** Coding interviews
- **Class 22:** Code reviews
- [Class 23: Introduction to Python](#)

## **Grading**

Grades will be 5% course feedback assignments, 5% participation, 30% in-class assignments, and 60% homework assignments (evenly split between all the homework assignments, including the final project). In-class assignments will be 50% correctness and 50% participation (awarded for at least 50% correctness). Assignments will report their percentage completion through the auto-grader, with points given for completing objectives and points taken away for things like memory errors or inconsistent style. Final grades will be curved if necessary.

## **Course feedback**

Several times over the semester, we will ask students to submit their anonymous feedback on the course. As a completely new course, we want to gauge the effectiveness of the course setup, assignments, and teaching style.

## **Participation**

During every class session we will have some “clicker”-type question to verify attendance. We want everyone to come to class so that you can support the other people at your table/group. Although assignments are individual and you shouldn't write code for anyone else, the class will be better for everyone if we can give advice and support and aid to each other. Also, if you get ahead of the in-class assignments, please start working on the homework!

## **Late Policy**

For in-class work, the two lowest scores for individual in-class assignment problems will be dropped. If you anticipate missing a class day, you are encouraged to complete that day's assignments beforehand.

For homework, over all the homework assignment problems, 48 total cumulative hours of tardiness are “free”. After this, each hour an assignment is late (rounded up by ceiling) will reduce its maximum score by one percentage point (so 80% completion of an assignment 10 hours late would be  $80\% * 90\% = 72\%$ ). The auto-grader will report these percentage calculations and keep

your highest final score from any submission. The 48 free hours of allowed homework tardiness will be applied at the end of the semester to maximize your final grade.

At any point, run `par-check` in a problem folder to see the highest score the auto-grader has recorded for you.

## Accessing CAEN Computers

To have a consistent development environment for all users, we will be using the CAEN computers in our classroom space. No matter which computer you log into, you will have access to your files. You are free to also use a personal computer if you like, although it may or may not be as easy to setup as the CAEN computers.

You can also access the CAEN computers remotely from your own computer through the `ssh` tool:

- While on campus run the following with your unqid and enter your password and DuoMobile 2-factor authentication:

```
ssh -X unqid@oncampus-course.engin.umich.edu
```

- When off campus, instead use:

```
ssh -X unqid@login-course.engin.umich.edu
```

- The `-X` option enables X11 Forwarding, which lets you open graphical programs over `ssh` and have them appear on your host computer. If you are running Windows as your host computer, you will then also need to install an X11 server to actually manage these windows. I recommend [xming](#). If you don't need to use a gui over `ssh`, you can omit this option. X11 Forwarding is pretty slow, though, so I don't recommend it for general work.

## Academic Honesty

The programs you submit, for both in-class and homework assignments, must be your own work, and significant similarity to other submissions will be considered highly suspect. Ultimately, though, the basic guideline is to be reasonable.

While working on problems, you are encouraged to search the internet to learn how to perform specific functions or techniques. In general, if you find a trivial one-liner on StackOverflow, you do not need to cite this. If you are copying a full algorithm, say for quicksort, you would need to cite this (or just use the standard library function `qsort!`). If that algorithm is a core objective of the assignment, however, then this would not be appropriate regardless of

citation. Especially when you implement trickier algorithms or mathematical calculations that you found somewhere online, it can be wise to include a link to the original description of that method in a comment. This makes it easier to check or resume your work later.

You are especially encouraged to get help from your peers! This means that after trying to figure out a problem or fix your code, please talk to other students. If you want them to look at your code, make sure they have already finished that section. Ask them for pointers about where the error is or what concepts or techniques to review, especially debugging techniques. Keep the conversation high-level and don't give or receive guided instructions on exactly what code to write. The most useful thing would be to point out flawed logic and allow the other student to come up with the fix themselves. For earlier brainstorming of problem solutions, discuss problems using a whiteboard or a sheet of paper so that everyone can still write their code for themselves. You should not show your own working code to another student who is struggling to complete theirs.

If on the homework you get significant help from your peers, please consider adding a comment in your code at the top of the file saying who you collaborated with and what information was shared. This may help avoid potential confusion in similar solutions. However, since sharing of code is not permitted, we still expect the small details to be significantly different.

If it has been determined that students have flagrantly violated this policy, we reserve the right to respond severely.

University of Michigan  
 Fall 2019 Instructor Report With Comments  
 ROB 599-001: Special Topics ROB  
 Acshi Haggenmiller

24 out of 26 students responded to this evaluation.

**Responses to the University-wide questions about the course:**

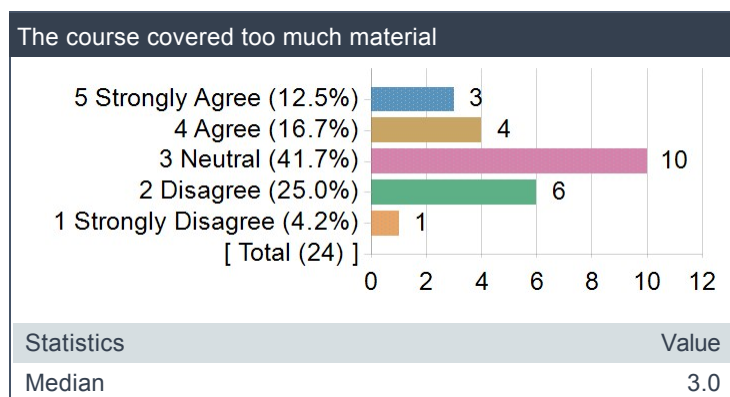
	SA	A	N	D	SD	N/A	Your Median	University-Wide Median	School/College Median
This course advanced my understanding of the subject matter. (Q1631)	23	1	0	0	0	0	5.0	4.5	4.6
My interest in the subject has increased because of this course.(Q1632)	19	3	1	1	0	0	4.9	4.2	4.5
I knew what was expected of me in this course.(Q1633)	14	7	3	0	0	0	4.6	4.4	4.5
Overall, this was an excellent course.(Q1)	21	3	0	0	0	0	4.9	4.2	4.5
I had a strong desire to take this course.(Q4)	15	7	2	0	0	0	4.7	4.0	4.5
As compared with other courses of equal credit, the workload for this course was...(SA=Much Lighter to SD=Much Heavier) (Q891)	3	0	6	13	2	0	2.3	3.0	3.0

**Responses to University-wide questions about the instructor:**

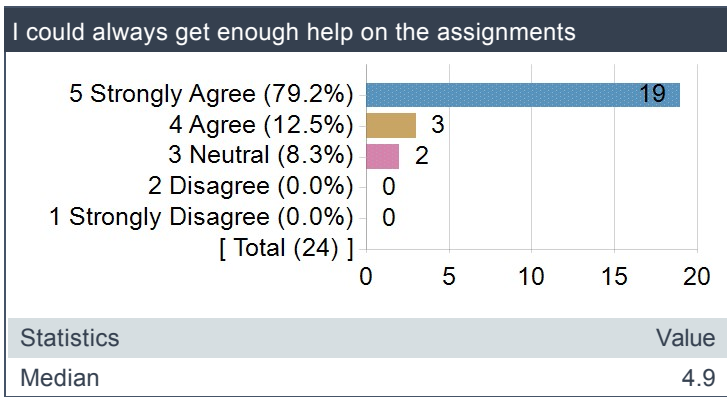
	SA	A	N	D	SD	N/A	Your Median	University-Wide Median	School/College Median
Overall, Acshi Haggenmiller was an excellent teacher. (Q2)	21	3	0	0	0	0	4.9	4.6	4.6
Acshi Haggenmiller seemed well prepared for class meetings.(Q230)	23	0	0	0	0	0	5.0	4.8	4.8
Acshi Haggenmiller explained material clearly.(Q199)	19	5	0	0	0	0	4.9	4.6	4.6
Acshi Haggenmiller treated students with respect. (Q217)	20	2	2	0	0	0	4.9	4.8	4.8

The medians are calculated from Fall 2019 data. University-wide medians are based on all UM classes in which an item was used. The school/college medians in this report are based on classes that are graduate level with enrollment of 16 to 74 in College of Engineering.

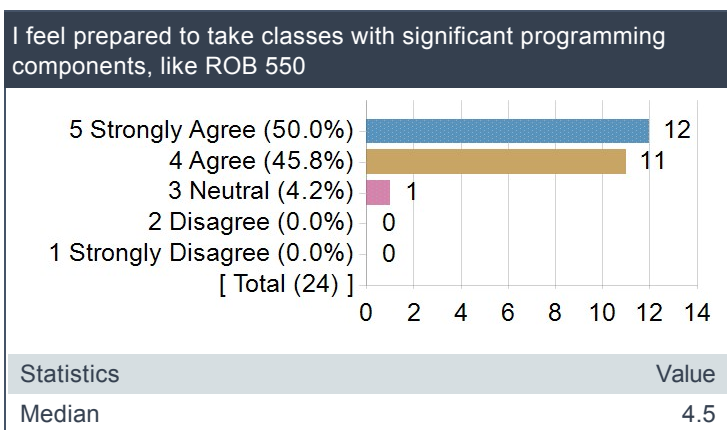
**Instructor-added question:**



**Instructor-added question:**



**Instructor-added question:**



## Written Comments

### Comment on the quality of instruction in this course.

#### Comments

Well. I am extremely appreciate his effort both in class and out of the lecture in terms of helping us out. He is a responsible person and capable of knowledge to resolve the programming issues for our classmates. However, I'd like to points out one of the tiny defect in his teaching. Cause this is a programming course and all the assignments are evaluated through auto-grader, plus he is carrying on this course all along. I know this could be overwhelming for him. But some of his test case is trivial to me. For an instance, there are dozens of ways to resolve a question and even different individuals can write a same algorithm in different ways. In that case, some part of our code is functioning correct in terms of behavior. But almost a tiny difference in the coding order might change our output in which case his auto-grader just thinks that ours are wrong. Myself have spent a lot time which is not necessary to find out where are the differences. That part is trivial and time-wasting which do nothing good to our understanding of programming and logic.

All in all, he is a nice person. I believe all of my classmates enjoy his lectures.

Acshi is an amazing instructor! 100% agree with and love his teaching methods. He is incredibly knowledgeable and even more helpful. He goes above and beyond to ensure we learn, by staying extra hours, promptly replying on Piazza, holding long office hours and more. Additionally, he designs interesting and creative hw problems that motivate me to program. Acshi's dedication to this course is unmatched, and he has set a very high bar for future instructors. Thank you!

Very very good, one of the best

Acshi did an outstanding job of dividing the time between lecturing and actually letting us do things. This combination is really the kind you need to learn to "think" programmatically. Overall, I've not only learnt the subject material but i've Also learnt how to teach.

It's a very good class and Acshi always taught materials clearly. He made great web pages and those are really helpful.

Very good and thorough.

Was very patient with students during class and office hours/

Responsive in piazza.

Given Acshi is teaching for the first time, he did an excellent excellent job. He was always patient and understanding and tried to help everyone who asked for it. Which is why it makes sense why this class size was so small.

Acshi showed a lot of effort in running this course, acting as professor and GSI. The content was well prepared.

Acshi was helpful during office hours, but sometimes his responses to questions were a bit harsh. I wish he'd started with the assumption that I'd already given the material my best effort. Sometimes it didn't feel like this was the case. I think it would be helpful to remember that many people in this class don't come from a strong programming background, and that we're trying our best.

Very knowledgeable, always there to help, listens to students for feedback. Covered a variety of topics incorporated with very interesting problem statements.

Most of the materials covered in this course are quite good. One problem is about the assignment. Admittly that it is hard to design and produce the assignments and Acshi did really a great job. It would be better to adjust the scoring criteria on autograder so that I could spent less time on debugging.

very good

Great.

This course would be hard to teach without someone as quality as quality as Acshi.

Perfect



**Course Approval Request Form**  
Office of the Registrar, University of Michigan

1210 USA Building  
500 S. State Street  
Ann Arbor, MI 48109-1382  
Phone: 734.763.2113  
Fax: 734.936.3148  
ro.curriculum@umich.edu  
ro.umich.edu

CHECK APPROPRIATE BOXES FOR ALL CHANGES

Action Requested

- New Course
- Modification of Existing Course
- Deletion of Existing Course

Date of Submission: 2019-11-04  
Effective Term: Fall 2020

<input checked="" type="checkbox"/>	<b>Course Offered</b> <input checked="" type="checkbox"/> Indefinitely <input type="checkbox"/> One term only	<b>RO USE ONLY</b> Date Received: Date Completed: Completed By:
-------------------------------------	---	--

**CURRENT LISTING**

**REQUESTED LISTING**


<input checked="" type="checkbox"/>	Dept (Home): Subject: Catalog:	Dept (Home): Robotics Subject: ROB Catalog: 511		
<input type="checkbox"/>	<input type="checkbox"/> Course is Cross-Listed with Other Departments			
	Department	Subject	Catalog Number	
<input checked="" type="checkbox"/>	Course Title (full title)	Course Title (full title) Robot Operating Systems		
<input checked="" type="checkbox"/>	Abbreviated Title (20 char)	Abbreviated Title (20 char) RobotOpSys		
<input checked="" type="checkbox"/>	Course Description (Please limit to 50 words and attach separate sheet if necessary) The Robot Operating Systems course provides an introduction to computational models, algorithms, and software systems for autonomous robot control that generalizes across a wide variety of machines. Topics covered include path and motion planning, reactive control, forward and inverse kinematics, numerical integration for dynamics, and robot middleware [design]. Significant programming.			
<input checked="" type="checkbox"/>	Full Term Credit Hours		Half Term Credit Hours	
	Undergraduate Min: 3	Graduate Min: 3	Undergraduate Min:	Graduate Min:
	Undergraduate Max: 3	Graduate Max: 3	Undergraduate Max:	Graduate Max:
<input checked="" type="checkbox"/>	Course Credit Type Undergraduate Student, Rackham Graduate Student			
<input type="checkbox"/>	Repeatability			
	<input type="checkbox"/> Course is Repeatable for Credit		<input type="checkbox"/> Course is Y graded	
	Maximum number of repeatable credits:		<input type="checkbox"/> Can be taken more than once in the same term	

Subject:      Catalog:	
<input checked="" type="checkbox"/>	<b>Grading Basis</b> <input checked="" type="checkbox"/> Graded (A – E) <input type="checkbox"/> Credit/No Credit <input type="checkbox"/> Satisfactory/Unsatisfactory <input type="checkbox"/> Pass/Fail <input type="checkbox"/> Business Administration
	<b>Add Consent</b> <input type="checkbox"/> Department Consent <input type="checkbox"/> Instructor Consent <input checked="" type="checkbox"/> No Consent
	<b>Drop Consent</b> <input type="checkbox"/> Department Consent <input type="checkbox"/> Instructor Consent <input checked="" type="checkbox"/> No Consent
	<b>Grading</b> <input type="checkbox"/> Not for Credit <input type="checkbox"/> Not for Degree Credit <input type="checkbox"/> Degree Credit Only

	CURRENT LISTING	REQUESTED LISTING
<input checked="" type="checkbox"/>	Advisory Prerequisite (254 char)	Advisory Prerequisite (254 char) Linear Algebra (Math 214,217,417,419 or equivalent) and Programming (EECS 280, 402 or equivalent)
<input checked="" type="checkbox"/>	Enforced Prerequisite (254 char) Minimum grade requirement:	Enforced Prerequisite (254 char) Minimum grade requirement: C
<input type="checkbox"/>	Credit Exclusions	Credit Exclusions
<input checked="" type="checkbox"/>	<b>Course Components</b> <input checked="" type="checkbox"/> Lecture <input type="checkbox"/> Seminar <input type="checkbox"/> Recitation <input type="checkbox"/> Lab <input checked="" type="checkbox"/> Discussion <input type="checkbox"/> Independent Study	<b>Graded Component</b> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>
		<b>Terms Typically Offered</b> <input checked="" type="checkbox"/> Fall <input type="checkbox"/> Winter <input type="checkbox"/> Spring <input type="checkbox"/> Summer <input type="checkbox"/> Spring/Summer
Cognizant Faculty Member Name: Chad Jenkins		Cognizant Faculty Member Title: Professor

**SIGNATURES ARE REQUIRED FROM ALL DEPARTMENTS INVOLVED (Please Print AND Sign Name)**

Contact Person: Denise Edmund      Email: dledmund@umich.edu      Phone: 7-2970

Curriculum Committee Member:	Print:	Date:
Curriculum Committee Chair:	Print:	Date:
Home Department Chair: 	Print: Jesse Grizzle	Date: 12/13/19
Cross-Listed Department Chair:	Print:	Date:
Cross-Listed Department Chair:	Print:	Date:
Cross-Listed Department Chair:	Print:	Date:



**DEPARTMENTAL/COLLEGE USE ONLY**

**Current:**

**Requested:**

Course Description

Course Description

The Robot Operating Systems course provides an introduction to computational models, algorithms, and software systems for autonomous robot control that generalizes across a wide variety of machines. Topics covered include path and motion planning, reactive control, forward and inverse kinematics, numerical integration for dynamics, and robot middleware [design]. Significant programming.

Class Length

Class Length

Full term

Contact hours (lecture):

Contact hours (lecture):

3

Contact hours (recitation)

Contact hours (recitation)

Contact hours (lab)

Contact hours (lab)

**Additional Info:**

Submitted by:

Home dept

Describe how this course fits with the degree requirements:

Satisfies the Acting or Reasoning requirement for the Masters and Doctoral Robotics Programs

ABET departmental program outcomes for undergraduate courses:

1,2,3,4,5,6,7

Special resources of facilities required for this course:

Supporting statement:

This Robot Operating Systems course aims to fill a need in the Robotics Program for breadth coverage over computational dimensions at the foundations of autonomous robotics. This need has developed from our longstanding teaching of mechanics-oriented robot modeling and control (as for the crosslisted ME 567/ROB 510). From the growth of the field of robotics, the computational core of robot modeling and control has emerged into its own right as an area of critical significance. Further, the proposed Robot Operating Systems course is intended to provide a pathway into further

study into autonomous robotics that can lead up and/or complement other computational and algorithmic course in robotics across the College. Complementary courses include those for mechanics-focused (ME 567/ROB 510), robot perception (e.g., EECS 568), planning (e.g., EECS 598) and dynamics (e.g., ME 543), artificial intelligence (e.g., EECS 592). This proposed course also extends and generalizes concepts only touched upon in the core ROB 550 Robotics Systems Laboratory course.



UNIVERSITY OF  
MICHIGAN

# Fall 2018 Instructor Report of MECHENG 567 001 - EECS 398 001 - EECS 567 001 - MFG 567 001 - ROB 510 001 for Odest Jenkins

Project Title: **Central Campus Fall 2018 Evaluation**

Course Audience: **73**

Responses Received: **36**

Response Ratio: **49.3%**

---

## Report Comments

This report is a summary that tabulates all quantitative ratings on a single page. Ratings are from the Fall 2018 teaching evaluations of MECHENG 567 001 - EECS 398 001 - EECS 567 001 - MFG 567 001 - ROB 510 001.

---

Prepared by: **Office of the Registrar**

Creation Date: **Tue, Jan 01, 2019**

## Responses to the University-wide questions about the course:

	SA	A	N	D	SD	N/A	Your Median	University-Wide Median	School/College Median
This course advanced my understanding of the subject matter.	26	9	1	0	0	0	4.8	4.5	4.7
My interest in the subject has increased because of this course.	23	10	1	2	0	0	4.7	4.1	4.4
I knew what was expected of me in this course.	14	15	6	1	0	0	4.2	4.4	4.5
Overall, this was an excellent course.	23	9	3	1	0	0	4.7	4.2	4.5
I had a strong desire to take this course.	20	15	1	0	0	0	4.6	4.0	4.4
As compared with other courses of equal credit, the workload for this course was... (SA=Much Lighter to SD=Much Heavier)	1	2	20	7	6	0	2.8	3.0	3.0

## Responses to the University-wide questions about the instructor:

	SA	A	N	D	SD	N/A	Your Median	University-Wide Median	School/College Median
Overall, Odest Jenkins was an excellent teacher.	22	12	1	1	0	0	4.7	4.5	4.6
Odest Jenkins seemed well prepared for class meetings.	24	11	0	1	0	0	4.8	4.8	4.7
Odest Jenkins explained material clearly.	18	11	4	2	0	1	4.5	4.6	4.6
Odest Jenkins treated students with respect.	30	5	1	0	0	0	4.9	4.8	4.8

The medians are calculated from Fall 2018 data. University-wide medians are based on all UM classes in which an item was used. The school/college medians in this report are based on classes that are graduate level with enrollment of 16 to 74 in College of Engineering.

University of Michigan  
 Fall 2019 Instructor Report With Comments  
 MECHENG 567 001 - EECS 367 001 - EECS 567 001 - MFG 567 001 - ROB 510  
 001  
 Odest Jenkins

32 out of 70 students responded to this evaluation.

**Responses to the University-wide questions about the course:**

	SA	A	N	D	SD	N/A	Your Median	University- Wide Median	School/College Median
This course advanced my understanding of the subject matter. (Q1631)	18	10	2	0	1	0	4.6	4.5	4.6
My interest in the subject has increased because of this course.(Q1632)	20	9	2	0	1	0	4.7	4.2	4.5
I knew what was expected of me in this course.(Q1633)	18	10	4	0	0	0	4.6	4.4	4.5
Overall, this was an excellent course.(Q1)	19	11	0	2	0	0	4.7	4.2	4.5
I had a strong desire to take this course.(Q4)	17	9	2	1	0	0	4.6	4.0	4.5
As compared with other courses of equal credit, the workload for this course was...(SA=Much Lighter to SD=Much Heavier) (Q891)	3	4	17	7	1	0	3.0	3.0	2.7

**Responses to University-wide questions about the instructor:**

	SA	A	N	D	SD	N/A	Your Median	University-Wide Median	School/College Median
Overall, Odest Jenkins was an excellent teacher.(Q2)	23	6	1	1	1	0	4.8	4.6	4.6
Odest Jenkins seemed well prepared for class meetings.(Q230)	23	5	3	1	0	0	4.8	4.8	4.7
Odest Jenkins explained material clearly.(Q199)	19	10	0	2	1	0	4.7	4.6	4.6
Odest Jenkins treated students with respect.(Q217)	28	4	0	0	0	0	4.9	4.8	4.8

The medians are calculated from Fall 2019 data. University-wide medians are based on all UM classes in which an item was used. The school/college medians in this report are based on classes that are graduate level with enrollment of 75 or greater in College of Engineering.

## Written Comments

### Comment on the quality of instruction in this course.

Comments
Great teacher and well content
Great instruction. OCJ is a very clear instructor, his lectures have a lot of examples.
I greatly enjoyed this course, the material and projects were interesting and helped me develop a good sense of the big picture tools being used in robot simulation and controls. I would have enjoyed a bit more of a focus on the dynamics of robots (for example the Manipulator equations) and there were times when the GSI and the Professor were not on the same page in terms of what was expected in an assignment, but overall it was still an incredibly enjoyable course.
The lectures went well, but they are somewhat broad. This leads to the real learning coming from doing the homework assignments. Because the assignments, for the most part, build off of each other, getting feedback on each assignment quickly would have been very helpful.
It would be better to accelerate the grading process, so we can manage this course better.
This is simply awesome, excellent, man
The tuition for each credit is really high. Averagely, it is really expensive for each course. However, a lot of classes were cancelled or changed as extended office hours. Michigan time was still applied, but the class still ends 10 mins ahead of time. During class, the lecture went really fast. Student were left few time to think. With some jokes in the class, the time left for substantial content was few. Now it is the end of this semester, but the second assignment has not been graded yet.
Excellent class. The structure of code matches with the overall progress of the class. it will be better if more TA are available for the class because a lot of time is wasted during waiting.
top notch
I wish we could go more in–depth to some of the concepts taught in class. That would make project implementation alot easier.
The professor has not graded an assignment in over 2 months. He has graded 1 of 5 submitted assignments. It is impossible to know how you are doing in the class and due to his delay in feedback, students are unable to submit regrades as promised in the syllabus. The lectures are often too high level and frequently tangential to a point that they are useless. I felt like I learned more from reading the text book than going to class. The TA's lab section was extremely useful.
The grade for assignments comes out really late. Grades start from assignment 2 didn't come out until the end of the semester. In this case students can not get feedback frequently from last homework, and this also increase the workload of students during finals.

# AutoRob

Introduction to Autonomous Robotics  
Michigan EECS 367

Robot Kinematics and Dynamics  
Michigan ME 567 EECS 567 ROB 510

Fall 2019



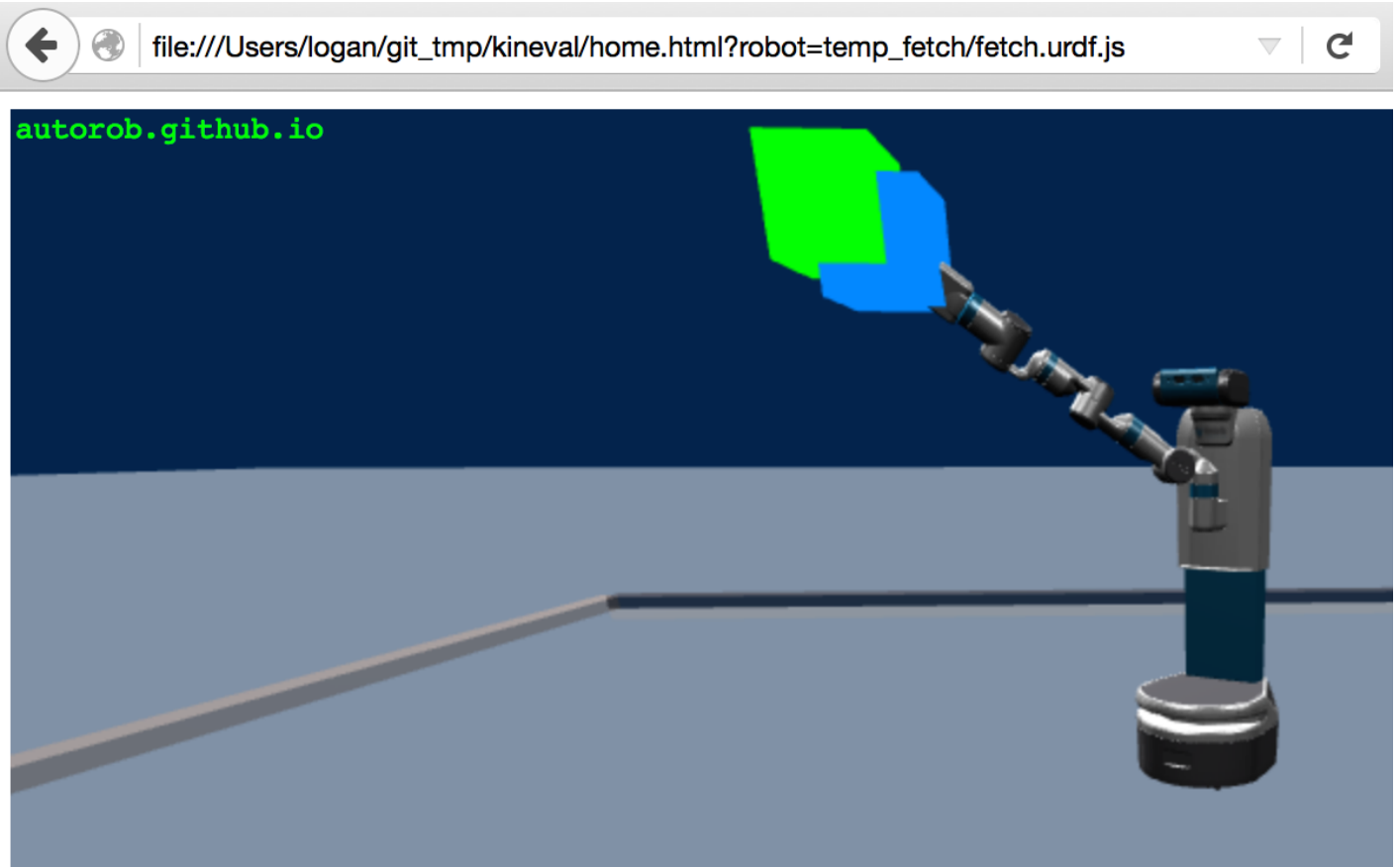
# Introduction

The AutoRob course provides an introduction to core topics in the modeling and control of autonomous robots. AutoRob has two sections: an undergraduate section offered as Introduction to Autonomous Robotics (EECS 367) and a graduate section offered as Robot Kinematics and Dynamics (ME/EECS 567 or ROB 510) with expanded advanced material. The AutoRob course can be thought of as the foundation to build "brains for robots". That is, given a robot as a machine with sensing, actuation, and computation, how do we build computational models, algorithms, and program that allow the robot to function autonomously? Such computation involves functions for robots to perceive the world, make decisions towards achieving a given objective, and transforming decided actions into motor commands. These functions are essential for modern robotics, especially for mobile manipulators such as the pictured [Fetch](#) robot.

The AutoRob course focuses on the issues of modeling and control for autonomous robots with an emphasis on manipulation and mobility. Successful completion of AutoRob will result in code modules for "mobile pick-and-place". That is, given a robot and perception of the robot's environment, the resulting code modules can enable the robot to pick up an object at an arbitrary location and place the object in a new location.

AutoRob projects ground course concepts through implementation in [JavaScript/HTML5](#) supported by the [KinEval code stencil](#) (snapshot below from Mozilla Firefox). These projects will cover graph search path planning ([A\\* algorithm](#)), basic physical simulation ([Lagrangian dynamics](#), [numerical integrators](#)), proportional-integral-derivative ([PID](#)) control, forward kinematics (3D geometric [matrix transforms](#), matrix stack composition of transforms, axis-angle rotation by [quaternions](#)), inverse kinematics ([gradient descent](#) optimization, geometric [Jacobian](#)), and motion planning (simple [collision detection](#), sample-based [motion planning](#)). Additional topics that could be covered include potential field navigation, Bayesian filtering, Cyclic Coordinate Descent, Monte Carlo localization, and Newton-Euler dynamics.

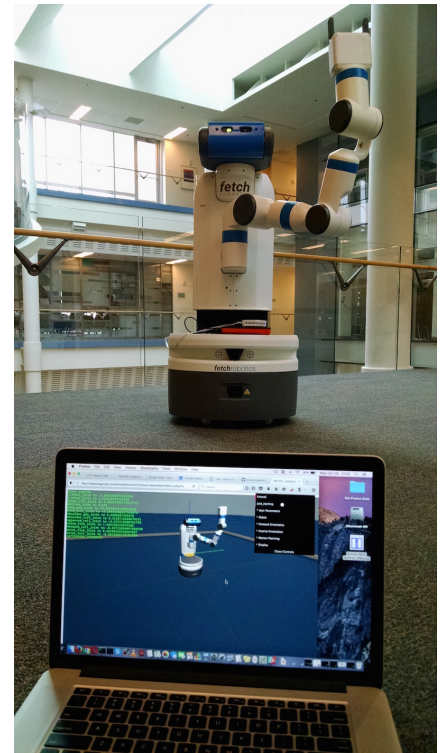




AutoRob projects will roughly follow conventions and structures from the [Robot Operating System \(ROS\)](#) and [Robot Web Tools \(RWT\)](#) software frameworks, as widely used across robotics. These conventions include the [URDF](#) kinematic modeling format, [ROS topic](#) structure, and the [rosbridge protocol](#) for [JSON](#)-based messaging. KinEval uses [threejs](#) for in-browser 3D rendering and [Numeric Javascript](#) for select matrix routines. Auxiliary code examples and stencils will often use the [jsfiddle](#) development environment.

### **You will use an actual robot (at least once)!**

KinEval allows for your code to work with any robot that supports the [rosbridge protocol](#), which includes any robot running ROS. Given a URDF description, the code you produce for AutoRob will allow you to view and control the motion of any mobile manipulation robot with rigid links. Your code will also be able to access the sensors and other software services of the robot for your continued work as a roboticist.



## Related Courses

AutoRob is a computing-friendly pathway into robotics, but does not cover the whole of robotics. The scope of AutoRob is robot modeling and control, which is well-suited as preparation for a Major Design Experience in EECS 467 (Autonomous Robotics Laboratory). EECS 467 and ROB 550 (Robotics Systems Laboratory) provides more extensive hands-on experience with a small set of real robotic platforms. In contrast, AutoRob emphasizes the creation of a generalized modeling and control software stack in simulation, with interfaces to work with a diversity of real robots.

AutoRob provides a computation-focused branch of ME 567 (Robot Kinematics and Dynamics). The common ME 567 is a more in-depth mathematical analysis of dynamics and control with extensive use of Denavit-Hartenberg parameters for kinematics. AutoRob spends more coverage on path and motion planning with use of quaternions and matrix stacks for kinematics. In AutoRob, [coding is believing](#).

AutoRob is also a complement to courses covering perception (EECS 568 Mobile Robotics, EECS 442 Computer Vision), robot building (EECS 498 Hands-on Robotics, ME 552 Mechatronics), robot simulation (ME 543 Analytical and Computational Dynamics), controls systems (EECS 460 Control Systems Analysis and Design, EECS 461 Embedded Control Systems), and artificial intelligence (EECS 492

Introduction to Artificial Intelligence), as well as general graduate courses in robotics (ROB 501 Math for Robotics, ROB 550 Robotic Systems Laboratory).

### **Commitment to equal opportunity**

As indicated in the [General Standards of Conduct for Engineering Students](#), this course is committed to a policy of equal opportunity for all persons and does not discriminate on the basis of race, color, national origin, age, marital status, sex, sexual orientation, gender identity, gender expression, disability, religion, height, weight, or veteran status. Please feel free to contact the course staff with any problem, concern, or suggestion. We ask that all students treat each other with respect.

### **Accommodations for Students with Disabilities**

If you believe an accommodation is needed for a disability, please let the course instructor know at your earliest convenience. Some aspects of this course, the assignments, the in-class activities, and the way the course is usually taught may be modified to facilitate your participation and progress. As soon as you make us aware of your needs, the course staff can work with the [Services for Students with Disabilities](#) (SSD, 734-763-3000) office to help us determine appropriate academic accommodations. SSD typically recommends accommodations through a Verified Individualized Services and Accommodations (VISA) form. Any information you provide is private and confidential and will be treated as such. For special accommodations for any academic evaluation (exam, quiz, project), the course staff will need to receive the necessary paperwork issued from the SSD office by *September 30, 2019*.

### **Student mental health and well being**

The University of Michigan is committed to advancing the mental health and wellbeing of its students. If you or someone you know is feeling overwhelmed, depressed, and/or in need of support, services are available. For help, please contact [Counseling and Psychological Services](#) (CAPS) by phone at 734-764-8312, during and after hours, on weekends and holidays, or through its counselors physically located in schools on both North and Central Campus. You may also consult [University Health Service](#) (UHS, 734-764-8320) as well as its [services for alcohol or drug concerns](#).

There is also a more comprehensive [listing of mental health resources](#) available on and off campus.

## Course Staff

### Faculty Instructor

[Chad Jenkins](#)

ocj addrsign umich

Office: Beyster 3644

Office Hours: Monday 3-5pm, Tuesday 1-3pm

### Graduate Student Instructor

Anthony Pipari

topipari addrsign umich

Office Hours (Beyster 1637): Monday 3-5pm, Friday 12:30-2:30pm

## Meeting time/place

### Course Lecture

Monday and Wednesday 1:30-2:40 MMT (Michigan Mean Time)

EECS 1500

### Laboratory Section (EECS 367)

Friday 2:30-3:20 MMT (Michigan Mean Time)

EECS 1500

## Discussion channel

[The AutoRob slack team](#) will be used for course-related discussions and announcements. [Slack](#) is a cloud-hosted online discussion and collaboration system with functionality that resembles [Internet Relay Chat \(IRC\)](#). Slack clients are available for most modern operating systems as well as through the web.

Students enrolled in AutoRob will receive an invitation to the AutoRob slack team. Upon accepting this invitation, you should be automatically subscribed to 9 channels:

- the *#general* channel for general course discussion and administrivia,
- the *#random* channel for non-course items that are of general interest to the course and larger field of robotics,
- seven assignment channels for each class project (*#asgn1-pathplan*, *#asgn2-pendularm*, *#asgn3-fk*, *#asgn4-dance-fsm*, *#asgn5-ik*, *#asgn6-rrt*, *#asgn7-best-use*),
- the *#advanced-extensions* channel for discussion and announcement of graduate advanced extensions to each project.

*Actively engaging in course discussions is a great way to become a better roboticist.*

## Prerequisites

This course has recommended prerequisites for "Linear Algebra" and "Data Structures and Algorithms", or permission from the instructor.

*Programming proficiency:* EECS 281 (Data Structures and Algorithms), EECS 402 (Programming for Scientists and Engineers), or proficiency in data structures and algorithms should provide an adequate programming background for the projects in this course. Interested students should consult with the course instructor if they have not taken EECS 281, EECS 402, or its equivalent, but have some other notable programming experience.

*Mathematical proficiency:* Math 214, 217, 417, 419 or proficiency in linear algebra should provide an adequate mathematical background for the projects in this course. Interested students should consult with the course instructor if they have not taken one of the listed courses or their equivalent, but have some other strong background with linear algebra.

Recommended optional proficiency: Differential equations, Computer graphics, Computer vision, Artificial intelligence

The instructor will do their best to cover the necessary prerequisite material, but no guarantees. Linear algebra will be used extensively in relation to 3D geometric transforms and systems of linear equations. Computer graphics is helpful for under-the-hood understanding of threejs. Computer vision and AI share common concepts with this course. Differential equations are used to cover modeling of motion dynamics and inverse kinematics, but not explicitly required.

## **Textbook**

The AutoRob course is compatible with both the Spong et al. and Corke textbooks (listed below), although only one of these books is needed. Depending on individual styles of learning, one textbook may be preferable over the other. Spong et al. is the listed required textbook for AutoRob (as well as ME 567) and is supplemented with additional handouts. The Corke textbook provides broader coverage with an emphasis on intuitive explanation. A pointer to the Lynch and Park textbook is provided for an alternative perspective on robot kinematics that goes deeper into spatial transforms in exponential coordinates. Lynch and Park also provides some discussion and context for using ROS. This semester, AutoRob will not officially support the Lynch and Park book, but will make every effort to work with students interested in using this text.

### **Robot Modeling and Control**

Mark W. Spong, Seth Hutchinson, and M. Vidyasagar  
Wiley, 2005

[Available at Amazon](#)

### ***Alternate textbooks***

### **Robotics, Vision and Control: Fundamental Algorithms in MATLAB**

Peter Corke  
Springer, 2011

## **Modern Robotics: Mechanics, Planning, and Control**

Kevin M. Lynch, Frank C. Park  
Cambridge University Press, 2017

### ***Optional texts***

## **JavaScript: The Good Parts**

Douglas Crockford  
O'Reilly Media / Yahoo Press, 2008

## **Principles of Robot Motion**

Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun  
MIT Press, 2005

## **Projects and Grading**

The AutoRob course will assign 7 projects (6 programming, 1 oral) and 5 quizzes. Each project has been decomposed into a collection of features, each of which is worth a specified number of points. AutoRob project features are graded as “checked” (completed) or “due” (incomplete). Prior to being assigned, upcoming projects will have the status of "pending." In terms of workload, each project is expected to take approximately 4 hours of work on average (as a rough estimate). Each quiz will consist of 4 short questions that will be within the scope of previously graded projects. In other words, each quiz question should be readily answerable given knowledge from correctly completing projects on time.

Individual final grades are assigned based on the sum of points earned from coursework (detailed in subsections below). The timing and due dates for course projects and quizzes will be announced on an ongoing basis. All project work must be checked by the end of classes.

## **EECS 367: Introduction to Autonomous Robotics**

In the undergraduate section, each fully completed project is weighted as 12 points and each correctly answered quiz question is weighted as 1 point. Based on this sum

of points from coursework, an overall grade for the course is earned as follows: An "A" grade in the course is earned if graded coursework sums to 93 points or above; A "B" grade in the course is earned if graded coursework sums to 83 points or above; a "C" grade in the course is earned if graded coursework sums to 73 points or above. The instructor reserves the option to assign appropriate course grades with plus or minus modifiers.

## **ME 567 | EECS 567 | ROB 510: Robot Kinematics and Dynamics**

In the graduate section, each fully completed project is weighted as 18 points, each correctly answered quiz question is weighted as 1 point. Students in this section have the opportunity to earn 4 additional points through an advanced extension of a course project. Examples of advanced extensions include implementation of an LU solver for linear systems of equations, inverse kinematics by Cyclic Coordinate Descent, one additional motion planning algorithm, point cloud segmentation, and a review of a current research publication in robotics. Advanced extensions are due by the course final grading deadline, and do not need to be completed for the deadlines of each assignment.

Based on the sum of points earned from coursework, an overall grade for the course is earned as follows: An "A" grade in the course is earned if graded coursework sums to 135 points or above; A "B" grade in the course is earned if graded coursework sums to 120 points or above; a "C" grade in the course is earned if graded coursework sums to 105 points or above. The instructor reserves the option to assign appropriate course grades with plus or minus modifiers.

### **Project Rubric (tentative and subject to change)**

The following project features are planned for AutoRob this semester. Students enrolled in ME/EECS 567 will complete all features. Students in the undergraduate section are not expected to implement features for the graduate section.

<b>Points</b>	<b>Sections</b>	<b>Feature</b>
		Assignment 1: 2D Path Planning
4	All	Heap implementation
8	All	A-star search



2	Grad	BFS
2	Grad	DFS
2	Grad	Greedy best-first
		Assignment 2: Pendularm
4	All	Euler integrator
4	All	Velocity Verlet integrator
4	All	PID control
1	Grad	Verlet integrator
2	Grad	RK4 integrator
3	Grad	Double pendulum
		Assignment 3: Forward Kinematics
2	All	Core matrix routines
8	All	FK transforms
2	All	Joint selection/rendering
2	Grad	Base offset transform
4	Grad	New robot definition
		Assignment 4: Dance Controller
6	All	Quaternion joint rotation
2	All	Interactive base control
2	All	Pose setpoint controller
2	All	Dance FSM
2	Grad	Joint limits
2	Grad	Prismatic joints
2	Grad	Fetch rosbridge interface
		Assignment 5: Inverse Kinematics
6	All	Manipulator Jacobian
3	All	Gradient descent with Jacobian transpose
3	All	Jacobian pseudoinverse
6	Grad	Euler angle conversion
		Assignment 6: Motion Planning
4	All	Collision detection
2	All	2D RRT-Connect
6	All	Configuration space RRT-Connect
6	Grad	RRT-Star

## **Project Submission and Regrading**

Git repositories will be used for project implementation, version control, and submission for grading. The implementation of an individual project is submitted through an update to the *master* branch of your designated repository. Updates to the master branch must be committed and pushed prior to the due date for each assignment for any consideration of full credit. Your implementation will be checked out and executed by the course staff. Through your repository, you will be notified by the course staff whether your implementation of assignment features is sufficient to receive credit.

## **Late Policy**

**Do not submit assignments late.** The course staff reserves the right to not grade late submissions. The course instructor reserves the right to decline late submissions and/or adjust partial credit on regraded assignments.

If granted by the course instructor, late submissions can be graded for partial credit, with the following guidelines. Submissions pushed within two weeks past the project deadline will be graded for 80% credit. Submissions pushed within four weeks of the project deadline will be graded for 60% credit. Submissions pushed at any time before the semester project submission deadline (December 13, 2019) will be considered for 50% credit. As a reminder, the course instructor reserves the right to decline late submissions and/or adjust partial credit on regraded assignments.

## **Regrading Policy**

The regrading policy allows for submission and regrading of projects up through the final grading of projects, which will be December 13 for the Fall 2019 Semester. This regrading policy will grant full credit for project submissions pushed to your repository before the corresponding project deadline. If a feature of graded project is returned as not completed (or "DUE"), your code can be updated for consideration at 80% credit. This code update must be pushed to your repository within two weeks from when the originally graded project was returned. Regrades of projects updated beyond this two week window can receive at most 60% credit. The course staff will allow one regrade for each grading iteration.

## Final Grading

All grading will be finalized on December 13, 2019. Regrading of specific assignments can be done upon request during office hours. No regrading will be done after grades are finalized.

## Repositories

You are expected to provide a **private** git repository for your work in this course with the course instructor added as a read/write collaborator. If needed, the course staff can assist in the setup of an online git repository through providers such as [github](#) or [bitbucket](#). All Michigan Engineering students have access to an account on the internal EECS [gitlab](#) server.

There are many different tutorials for learning how to use git repositories. The AutoRob course site has its own basic [quick start tutorial](#). The EECS gitlab server has a basic [quick start tutorial](#). The [Pro Git book](#) provides an in-depth introduction to git and version control. As different people often learn through different styles, the [Git Magic tutorial](#) has also proved quite useful when a different perspective is needed. [git: the simple guide](#) has often been a great and accessible quick start resource.

We expect students to use these repositories for collaborative development as well as project submission. It is the responsibility of each student group to ensure their repository adheres to the Collaboration Policy and submission standards for each assignment. Submission standards and examples will be described for each assignment as needed.

**IMPORTANT:** do not modify the directory structure in the [KinEval](#) stencil. For example, the file "home.html" should appear in the top level of your repository. Repositories that do not follow this directory structure will not be graded.

## Code Maintenance Policy and Branching

This section outlines expectations for maintenance of source code repositories used by students for submission of their work in this course. Repositories that do not maintain these standards will not be graded at the discretion of the course staff.

Code submitted for projects in this course must reside in the top level directory of the *master* branch of your repository. The directory structure provided in the [KinEval](#) stencil must not be modified. For example, the file "home.html" should appear in the top level directory of your repository.

The *master* branch must always maintain a working (or stable) version of your code for this course. Code in the *master* branch can be analyzed at any time with respect to any assignment whose due date has passed. Improperly functioning code on the *master* branch can affect the grading of an assignment (even after the assignment due date) up to the start of the following semester.

The *master* branch must always be in compliance with the Michigan Honor Code and Michigan Honor License, as described below in the course Collaboration Policy. A commit of code to your *master* branch must be signed with your name and the instructor name at the bottom of the file named LICENSE with an unmodified version of the Michigan Honor License. Without a properly asserted license file, a code commit to your repository will be considered ineligible for grading as an incomplete submission.

If advanced extension features have been implemented and are ready for grading, such features must be listed in the file "advanced\_extensions.txt" in the top level directory of the *master*. Advanced extension features not listed in this file may not be graded at the discretion of the course staff.

## BRANCHING

You are encouraged to update your repository often with the help of branching. Branching allows you to spawn a copy of code in your *master* branch for development, and merging these changes back into *master* once successfully updated. For example, you can create an *Assignment-2* branch for your work on the second project, as it is in development and experimentation, while keeping your *master* branch stable for grading. Once you are confident in your implementation of the second project, you can merge your *Assignment-2* branch back into the *master* branch. The *master* branch at this point will have working stable versions of the first and second projects, which can be graded. Similarly, an *Assignment-3* branch can be

created for the next project as it develops, and then merged into the *master* branch when ready for grading. This configuration allows your work to be continually updated and build upon such that versions are tracked and interruptions due to grading are minimized.

## **Collaboration Policy**

This collaboration policy covers all course material and assignments unless otherwise stated. All submitted assignments for this course must adhere to the Michigan Honor License (the [3-Clause BSD License](#) plus two academic integrity clauses).

Course material, concepts, and documentation may be discussed with anyone. Discussion during and examination or quiz is not allowed with anyone other than a member of the course staff. Assignments may be discussed with the other students at the conceptual level. Discussions may make use of a whiteboard or paper. Discussions with others (or people outside of your assigned group) cannot include writing or debugging code on a computer or collaborative analysis of source code that is not your own. You may take notes away from these discussions, provided these notes do not include any source code.

The code for your implementation may not be shown to anyone outside of your group, including granting access to repositories or careless lack of protection. You do not need to hide the screen from anyone, but you should not attempt to show anyone your code. When you are done using any robot device such that another group may use it, you must remove all code you have put onto the device. You may not share your code with others outside of your group. At any time, you may show others the implemented program running on a device or simulator, but you may not discuss specific debugging details about your code while doing so.

This policy applies not only applies to collaboration during the current semester, but also any past or future instantiations of this course. Although course concepts are intended for general use, your implementation for this course must remain private after the completion of the course. It is expressly prohibited to share any code previously written and graded for this course with students currently enrolled in this

course. Similarly, it is expressly prohibited for any students currently enrolled in this course to refer to any code previously written and graded for this course.

**IMPORTANT:** To acknowledge compliance with this collaboration policy, append your name the file "LICENSE" in the main directory of your repository with the following text. This appending action is your attestation of your compliance with the Michigan Honor License and the Michigan Honor Code statement:

```
"I have neither given nor received unauthorized aid on this course project implementation, nor have I concealed any violations of the Honor Code."
```

This attestation of the honor code will be considered updated with the current date and time of each commit to your repository. Repository commits that do not include this attestation of the honor code will not be graded at the discretion of the course instructor.

Should you fail to abide by this collaboration policy, you will receive no credit for this course. The University of Michigan reserves the right to pursue any means necessary to ensure compliance. This includes, but is not limited to prosecution through The College of Engineering Honor Council, which can result in your suspension or expulsion from the University of Michigan. Please refer to the [Engineering Honor Council](#) for additional information.

## Course Schedule (tentative and subject to change)

Note: Assignment descriptions will have updated assignment due dates. Assignment due dates listed in the schedule are merely a guide.

Note: Updated lecture slides will be posted after the associated lecture has been given. Slides provided below are from a previous offering of this course, and provided as a courtesy.

Date	Topic	Reading	Project
Sep 4	<a href="#">Initialization</a> : Course overview, Robotics roadmap, Path planning quick start	Spong Ch.1 <hr/> Corke Ch.1	Setup git repository

[What is a robot?](#): Brief history and definitions for robotics

Out: [Path Planning](#)

**Sep 6** 367 Lab: [Git-ing started with git, JavaScript, and KinEval](#)

Week 2

**Sep 9** [Path Planning](#): DFS, BFS, A-star, Greedy best first

[Wikipedia](#)

**Sep 11** [JavaScript and git tutorial](#): Project workflow, JS/HTML5, Document Object Model, Data Structures in JS, Animation in JS/H5, Version Control

Crockford, [HTML Sandbox](#), [hello.html \(source\)](#), [JavaScript by Example \(source\)](#), [hello\\_anim \(source\)](#), [hello\\_anim\\_text \(source\)](#)

**Sep 13** 367 Lab: [heapsort.html and search\\_canvas.html code overview](#)

Week 3

**Sep 16** **Extended office hours**

**Sep 18** Quiz 1

Due: [Path Planning](#)  
Out: [Pendularm](#)

**Sep 20** 367 Lab: [pendularm1.html code overview](#)

Week 4

**Sep 23** [Dynamical Simulation](#): Simple pendulum, Lagrangian equation(s) of motion, Initial value problem, Explicit integrators: Euler, Verlet, and Runge-Kutta 4, Double pendulum

Spong Ch.7 | Corke Ch.9  
[Euler's Method](#)  
[Verlet Integration](#),  
[Runge-Kutta](#);  
Witkin&Baraff 1998: [Dynamics](#)  
Witkin&Baraff 1998: [Integrators](#)

**Sep 25** [Motion Control](#): Cartesian vs. generalized coordinates, open-loop vs. closed-loop control, PID control; Rigid body dynamics

Spong 6.3,  
[Vondrak+ 2012](#)

---

[Astrom Ch. 6](#)

**Sep 27** 367 Lab: [pendularm1.html support](#)

Week 5

**Sep 30** [Linear Algebra Refresher](#): Systems of linear equations, vector spaces and operations, least squares approximations

Spong A-B

---

Corke D

[Forward Kinematics](#): Kinematic chains, URDF, homogeneous transforms, matrix stack traversal, D-H convention

Spong 2, 3.1, 3.2

---

Corke 7.1-2

**Oct 2** [Axis-angle Rotation and Quaternions](#): Motors, Euler angles, gimbal lock, Rodrigues rotation,

[Handout 1, 2](#)  
[Daniilidis 1999](#)

Due: [Pendularm](#)  
Out: [Forward](#)

	rotation in complex spaces, Dual quaternions and screw coordinates	Corke 2.2-3	Kinematics
<b>Oct 4</b>	367 Lab: <a href="#">KinEval and urdf.js code overview</a>		
	Week 6		
<b>Oct 7</b>	Quiz 2		
<b>Oct 9</b>	<b>Course meeting cancelled</b> - off-site meeting		
<b>Oct 11</b>	367 Lab: Quaternions in KinEval		
	Week 7		
<b>Oct 14</b>	<b>No course meeting</b> - Fall Study Break		
<b>Oct 16</b>	<a href="#">Reactive Controllers</a> : Reactive and Deliberative Decision Making, Finite State Machines, Subsumption Architecture	<a href="#">Brooks 1986</a> , <a href="#">Mataric 1992</a> , <a href="#">Platt+ 2004</a> , <a href="#">Cunningham+ 2015</a>	Due: Forward Kinematics Out: <a href="#">Dance Contest</a>
<b>Oct 18</b>	367 Lab: <a href="#">KinEval pose parameters and HTML5 audio</a>		
	Week 8		
<b>Oct 21</b>	<a href="#">Robot Middleware</a> : Hardware Abstraction, ROS, LCM, Publish-subscribe messaging, rosbridge, Client-server messaging	<a href="#">Quigley+ 2009</a> , <a href="#">Huang+ 2010</a> , <a href="#">Toris+ 2015</a>	
<b>Oct 23</b>	<a href="#">Inverse Kinematics 1 - Closed-form</a> : Joint vs. Endeffector control, Planar 2-link arm, Closed form solutions, Cyclic Coordinate Descent	Spong 3.3 <hr/> Corke 7.3  <a href="#">IK robot game</a>	
<b>Oct 25</b>	367 Lab: <a href="#">rosbridge/FK: connect your code to a real robot</a>		
	Week 9		
<b>Oct 28</b>	<a href="#">Inverse Kinematics 2 - Non-linear Optimization</a> : Gradient descent, Manipulator Jacobian, Jacobian transpose and pseudoinverse	Spong 4, <a href="#">Wang&amp;Chen 1991</a> , <a href="#">Buss 2009</a> , <a href="#">Beeson+ 2015</a> <hr/> Corke 8	
<b>Oct 30</b>	Quiz 3		Due: Dance Contest Out: <a href="#">Inverse Kinematics</a>
<b>Nov 1</b>	367 Lab: <a href="#">KinEval IK control flow and parameters</a>		



Week 10

- Nov 4** [Bug Algorithms](#): Reaction vs. Deliberation revisited, Bug[0-2], Tangent Bug [Lumelsky+ 1986](#), [Kamon+ 1996](#)
- 
- Nov 6** [ROS/catkin tutorial session](#) Corke 5
- Nov 8** 367 Lab: ROS/catkin/rosbridge continued Example tutorial result

Week 11

- Nov 11** [Configuration Spaces](#): Curse of dimensionality, Configuration space vs. Workspace, Minkowski planning, Costmaps, Holonomicity Spong 5
- 
- Nov 13** [Sample-based Planning](#): Probabilistic roadmaps, RRT-based motion planning [Kavraki+ 1996](#), [Kuffner+ 2000](#), [McMahon+ 2018](#)
- [Potential fields](#): Gradient descent revisited, local search, downhill simplex, Wavefront planning [Khatib 1986](#), [Jarvis 1993](#), [Zelinsky 1992](#) Due: Inverse Kinematics  
Out: [Motion Planning](#)
- Nov 15** 367 Lab: search\_canvas.html revisited for 2D RRT

Week 12

- Nov 18** [Collision Detection](#): 3D Triangle-Triangle Testing, Oriented Bounding Boxes, Axis-Aligned Bounding Boxes, Separating Axis Theorem [Gottschalk+ 1996](#), [Moller 1997](#)
- Nov 20** Quiz 4
- Nov 22** 367 Lab: [KinEval RRT stencil and AABB collision detection](#)

Week 13

- Nov 25** Extended Office Hours -
- Nov 27** **Course meeting cancelled** - Thanksgiving Recess

Week 14

- Dec 2** 3D Point Cloud Segmentation: Point cloud segmentation, Principal Components Analysis, Connected components [PCA](#), [Rusu+ 2008](#), [ten Pas+ 2014](#),
- Task Planning Overview: Decision making revisited, declarative programming, axiomatic state, planning operators [Fikes+ 1971](#), [Laird+ 1987](#), [Traflet+ 2013](#), [Zeng+ 2018](#)

- Dec 4** Localization and Mapping Overview: Bayes rule, Bayesian filtering, Monte Carlo Localization, Factor Graphs, SGD-SLAM, Scene estimation [Dellaert+ 1999](#), [Olson+ 2006](#), [Sui+ 2017](#) Due: Motion Planning
- Week 15
- Dec 9** Quiz 5
- Dec 11** Due: Best Use of Robotics
- Dec 13** Grading finalized

Slides from this course borrow from and are indebted to many sources from around the web. These sources include a number of excellent robotics courses at various universities.

## Assignment 1: Path Planning

**Due 11:59pm, Wednesday, September 18, 2019**

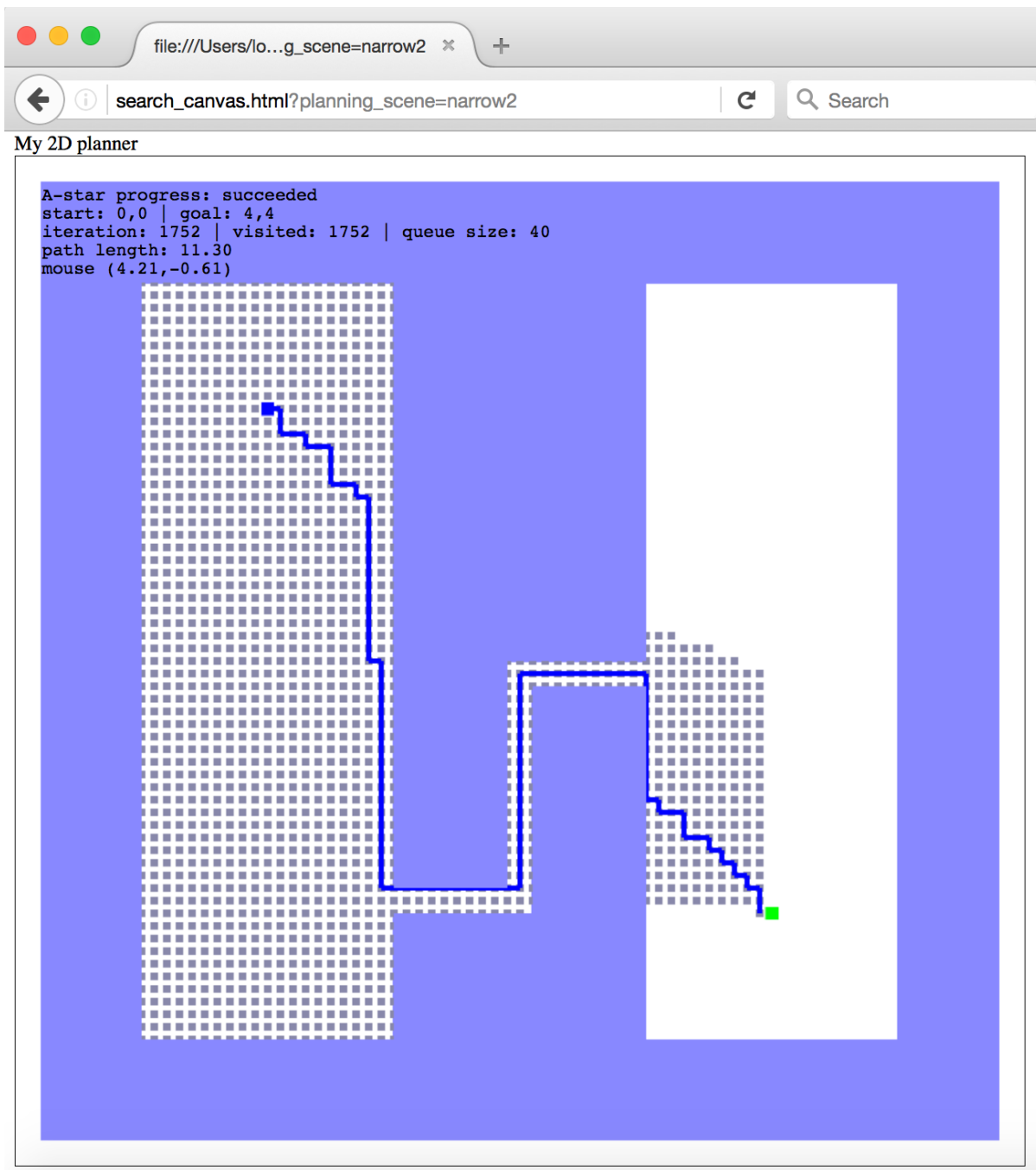
The objective of the first assignment is to implement a collision-free path planner in JavaScript/HTML5. Path planning is used to allow robots to autonomously navigate in environments from previously constructed maps. A path planner essentially finds a set of waypoints (or setpoints) for the robot to traverse and reach its goal location without collision. As covered in other courses (EECS 467, ROB 550, or EECS 568), such maps can be estimated through methods for [simultaneous localization and mapping](#). Below is an example from [EECS 467](#) where a [robot performs autonomous navigation](#) while simultaneously building an occupancy grid map:

## EECS 467 Winter 2017 escape challenge - team2



For this assignment, you will implement the planning part of autonomous navigation as an A-star graph search algorithm. A-star infers the shortest path from a start to a goal location in an arbitrary 2D world with a known map (or collision geometry). This A-star implementation will consider locations in a uniformly space grid. You will implement a heap data structure as a priority queue for visiting these locations.

If properly implemented, the A-star algorithm should produce the following path (or path of similar length) using the provided code stencil:



## Cloning the Stencil Repository

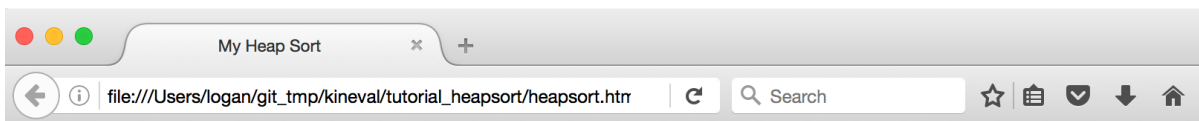
The first step for completing this project (and all projects for AutoRob) is to clone the [KinEval stencil repository](#). The appended git quick start below is provided those unfamiliar with git to perform this clone operation, as well as committing and pushing updates for project submission. **IMPORTANT:** the stencil repository should be cloned and **not forked**.

Throughout the KinEval code stencil, there are markers with the string "STENCIL" for code that needs to be completed for course projects.

## Heap Sort Tutorial

For those new to JavaScript/HTML5, the recommended starting point is to complete the heap sort implementation in the "tutorial\_heapsort" subdirectory of the stencil repository. In this directory, a code stencil in JavaScript/HTML5 is provided in two files: "heapsort.html" and "heap.js". Comments are provided throughout these files to describe the structure of JavaScript/HTML5 and its programmatic features. In addition, there are other tutorial-by-example files in the "tutorial\_js" directory. Any of these files can be run by simply opening them in a web browser.

Opening "heapsort.html" will show the result of running the incomplete heap sort implementation provided by the code stencil:



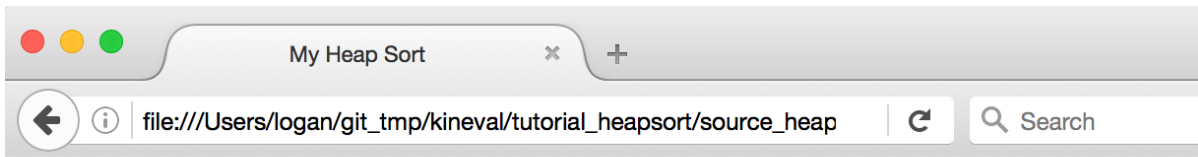
### My Heap Sort

check

numbers to sort: 1518 3544 102 7763 338 2893 5231 5687 3271 7759 5304 9224 5249 16 5499 835 4639 1000 9608 8602  
my heaping functions are not yet implemented

To complete the heap sort implementation, complete the heap implementation in heap.js at the locations marked "STENCIL". In addition, the inclusion of heap.js in the execution of the heap sort will require modification of heapsort.html.

A successful heap sort implementation will show the following result for a randomly generated set of numbers:



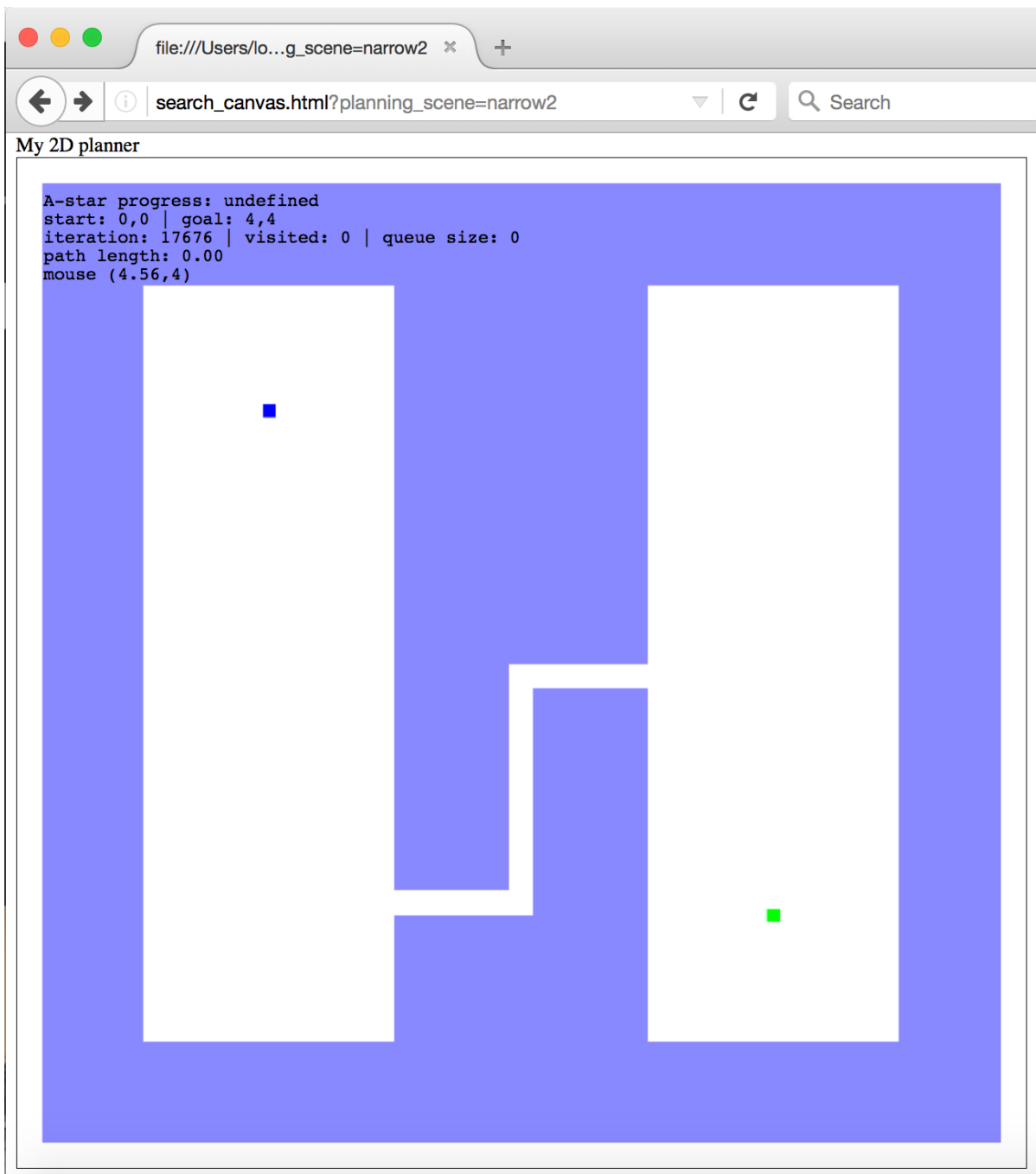
## My Heap Sort

8320 9772 5185 3865 6409 4631 797 3413 1472 319 8465 5949 3157 3911 4921 1165 2575

```
check
numbers to sort: 5949 8320 1472 6409 3865 4921 3157 459 5185 797 8465 3911 2575 100 1165 4631 9772 3413 78 319
heap (insert 5949): 5949
heap (insert 8320): 5949 8320
heap (insert 1472): 1472 8320 5949
heap (insert 6409): 1472 6409 5949 8320
heap (insert 3865): 1472 3865 5949 8320 6409
heap (insert 4921): 1472 3865 4921 8320 6409 5949
heap (insert 3157): 1472 3865 3157 8320 6409 5949 4921
heap (insert 459): 459 1472 3157 3865 6409 5949 4921 8320
heap (insert 5185): 459 1472 3157 3865 6409 5949 4921 8320 5185
heap (insert 797): 459 797 3157 3865 1472 5949 4921 8320 5185 6409
heap (insert 8465): 459 797 3157 3865 1472 5949 4921 8320 5185 6409 8465
heap (insert 3911): 459 797 3157 3865 1472 3911 4921 8320 5185 6409 8465 5949
heap (insert 2575): 459 797 2575 3865 1472 3157 4921 8320 5185 6409 8465 5949 3911
heap (insert 100): 100 797 459 3865 1472 3157 2575 8320 5185 6409 8465 5949 3911 4921
heap (insert 1165): 100 797 459 3865 1472 3157 1165 8320 5185 6409 8465 5949 3911 4921 2575
heap (insert 4631): 100 797 459 3865 1472 3157 1165 4631 5185 6409 8465 5949 3911 4921 2575 8320
heap (insert 9772): 100 797 459 3865 1472 3157 1165 4631 5185 6409 8465 5949 3911 4921 2575 8320 9772
heap (insert 3413): 100 797 459 3413 1472 3157 1165 4631 3865 6409 8465 5949 3911 4921 2575 8320 9772 5185
heap (insert 78): 78 100 459 797 1472 3157 1165 4631 3413 6409 8465 5949 3911 4921 2575 8320 9772 5185 3865
heap (insert 319): 78 100 459 797 319 3157 1165 4631 3413 1472 8465 5949 3911 4921 2575 8320 9772 5185 3865 6409
heap (extract 78): 100 319 459 797 1472 3157 1165 4631 3413 6409 8465 5949 3911 4921 2575 8320 9772 5185 3865
heap (extract 100): 319 797 459 3413 1472 3157 1165 4631 3865 6409 8465 5949 3911 4921 2575 8320 9772 5185
heap (extract 319): 459 797 1165 3413 1472 3157 2575 4631 3865 6409 8465 5949 3911 4921 5185 8320 9772
heap (extract 459): 797 1472 1165 3413 6409 3157 2575 4631 3865 9772 8465 5949 3911 4921 5185 8320
heap (extract 797): 1165 1472 2575 3413 6409 3157 4921 4631 3865 9772 8465 5949 3911 8320 5185
heap (extract 1165): 1472 3413 2575 3865 6409 3157 4921 4631 5185 9772 8465 5949 3911 8320
heap (extract 1472): 2575 3413 3157 3865 6409 3911 4921 4631 5185 9772 8465 5949 8320
heap (extract 2575): 3157 3413 3911 3865 6409 5949 4921 4631 5185 9772 8465 8320
heap (extract 3157): 3413 3865 3911 4631 6409 5949 4921 8320 5185 9772 8465
heap (extract 3413): 3865 4631 3911 5185 6409 5949 4921 8320 8465 9772
heap (extract 3865): 3911 4631 4921 5185 6409 5949 9772 8320 8465
heap (extract 3911): 4631 5185 4921 8320 6409 5949 9772 8465
heap (extract 4631): 4921 5185 5949 8320 6409 8465 9772
heap (extract 4921): 5185 6409 5949 8320 9772 8465
heap (extract 5185): 5949 6409 8465 8320 9772
heap (extract 5949): 6409 8320 8465 9772
heap (extract 6409): 8320 9772 8465
heap (extract 8320): 8465 9772
heap (extract 8465): 9772
heap (extract 9772):
sorted: 78 100 319 459 797 1165 1472 2575 3157 3413 3865 3911 4631 4921 5185 5949 6409 8320 8465 9772
```

## Graph Search Stencil

For the path planning implementation, a JavaScript/HTML5 code stencil has been provided in the file "search\_canvas.html" within the "project\_pathplan" subdirectory. Opening this file in a browser should display an empty 2D world displayed in an [HTML5 canvas](#) element.



There are five planning scenes that have been provided within this code stencil: "empty", "misc", "narrow1", "narrow2", and "three\_sections". The choice of `planning_scene` can be specified from the URL given to the browser, described in the usage in the file. For example, the URL `search_canvas.html?planning_scene=narrow2` will bring up the "narrow2" planning world. Other execution parameters, such as start and goal location, can also be specified through the document URL.

This code stencil is implemented to perform graph search iterations interactively in the browser. The core of the search implementation is performed by the function

iterateGraphSearch(). This function performs a graph search iteration for a single location in the A-star execution. The browser implementation cannot use a while loop over search iterations, as in the common A-star implementation. Such a while loop would keep control within the search function, and cause the browser to become non-responsive. Instead, the iterateGraphSearch() gives control back to the main animate() function, which is responsible for updating the display and user interaction.

Within the code stencil, you will complete the functions initSearchGraph() and iterateGraphSearch() as well as add functions for heap operations. Locations in "search\_canvas.html" where code should be added are labeled with the "STENCIL" string. In initSearchGraph() creates a 2D array over locations to be searched. Each element of this array contains various information computed by the search algorithm for a particular location. iterateGraphSearch() should use three of the provided functions to perform a search iteration. testCollision() returns a boolean of whether a given 2D location, as a two-element vector, is in collision with the planning scene. draw\_2D\_configuration() draws a square at a given location in the planning world to indicate that location has been explored. Once the search is complete, drawHighlightedPathGraph() will render the path produced by the search algorithm between a given location and the start location. The global variable search\_iterate should be set to false to end animation loop.

## **Graduate Section Requirement**

In addition to the A-star algorithm, students in the graduate section of AutoRob must additionally implement path planning by Depth-first search, Breadth-first search, and Greedy best-first search. An additional report, as file "report.html" in the "project\_pathplan" directory, is required that: 1) shows results from executing every search algorithm with every planning world for various start and goal configurations and 2) synthesizes these results into coherent findings about these experiments.

For effective communication, it is recommended to think of "report.html" like a short research paper: motivate the problem, set the value proposition for solving the problem, describe how your methods can address the problem, and show results the demonstrate how well these methods realize the value proposition. Visuals are highly recommended to complement this description. The best research papers can be read



in three ways: once in text, once in figures, and once in equations. It is also incredibly important to remember that writing in research is about generalizable understanding of the problem more than a specific technical accomplishment.

## ADVANCED EXTENSIONS

Advanced extensions can be submitted anytime before the final grading is complete. Concepts for several of these extensions will not be covered until later in the semester. Any new path planning algorithm must be implemented within the file "search\_canvas.html" under the "project\_pathplan" directory, and invoked through a parameter given through the URL. For example, the Bug0 algorithm must be invoked by adding the argument `?search_alg="Bug0"` to the URL. Thus, a valid invocation of Bug0 for the Narrow2 world could use the URL:

```
file:///project_pathplan/source_search_canvas.html?planning_scene=narrow2?  
search_alg="Bug0"
```

The same format must be used to invoke any other algorithm (such as Bug1, Bug2, TangentBug, Wavefront, etc.).

Of the 4 possible advanced extension points, two additional points for this assignment can be earned by implementing the "Bug0", "Bug1", "Bug2", and "TangentBug" navigation algorithms. The implementation of these bug algorithms must be contained within the file "search\_canvas.html" under the "project\_pathplan" directory.

Of the 4 possible advanced extension points, two additional points for this assignment can be earned by implementing navigation by "Potential" fields and navigation using the "Wavefront" algorithm. The implementation of these potential-based navigation algorithms must be contained within the file "search\_canvas.html" under the "project\_pathplan" directory.

Of the 4 possible advanced extension points, one additional point for this assignment can be earned by implementing a navigation algorithm using a probabilistic roadmap ("PRM"). This roadmap algorithm implementation must be contained within the file "search\_canvas.html" under the "project\_pathplan" directory.

Of the 4 possible advanced extension points, one additional point for this assignment can be earned by implementing costmap functionality using morphological operators. Based on the computed costmap, the navigation routine would provide path cost in addition path length for a successful search. The search implementation with this costmap must be contained within the file "search\_canvas\_costmap.html" under the "project\_pathplan" directory.

Of the 4 possible advanced extension points, one additional point for this assignment can be earned by implementing a priority queue through an [AVL tree](#) or a [red-black tree](#). The search implementation with this priority queue must be contained within the file "search\_canvas\_balancedtree.html" under the "project\_pathplan" directory.

## Project Submission

For turning in your assignment, ensure your completed project code has been committed and pushed to the *master* branch of your repository.

To ensure proper submission of your assignments, please do the following:

- confirm with the course instructor (ocj addrsign umich) with your name, email address, and pointer to your repository,
- ensure the instructor has push/admin access to your repository, which can be confirmed/addressed through email or office hours (or by seeing that the instructor has committed the file "grading.txt")

If you are paying attention, you should also add a directory to your repository called "me". This "me" directory should include a simple webpage in the file "me.html". The "me.html" file should have a title with your name, an h1 tag with your name, an img tag includes a picture of you from the file "me.png", body with a brief introduction about you, and a script tag that prints the result of `Array(16).join("wat"-1)+" Batman!"` to the console.

## Assignment 2: Pendularm

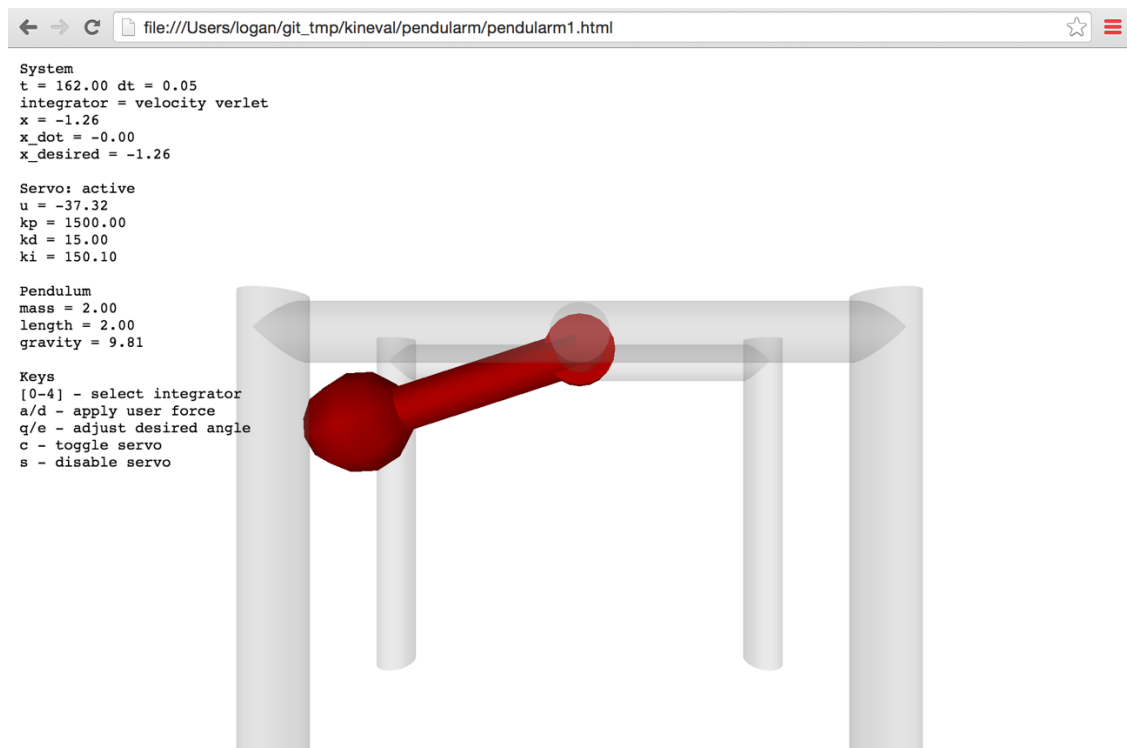
**Due 11:59pm, Wednesday, October 2, 2019**

Physical simulation is widely used across robotics to test robot controllers. Testing in simulation has many benefits, such as avoiding the risk of damaging a (likely expensive) robot and faster development of controllers. Simulation also allows for consideration of environments not readily available for testing, such as interplanetary

exploration (as in the example below for the NASA Space Robotics Challenge). We will now model and control our first robot, the Pendularm, to achieve an arbitrary desired setpoint state.



As an introduction to building your own robot simulator, your task is to implement a physical dynamics and servo controller for a simple 1 degree-of-freedom robot system. This system is 1 DOF robot arm as a frictionless [simple pendulum](#) with a rigid massless rod and idealized motor. A visualization of the Pendularm system is shown below. Students in the graduate section will extend this system into a 2-link 2-DOF robot arm, as an actuated [double pendulum](#).



The code stencil for the Pendularm assignment is available within the "pendularm" subdirectory of KinEval in the file [pendularm1.html](file:///Users/logan/git_tmp/kineval/pendularm/pendularm1.html).

For physical simulation, you will implement several numerical integrators for a pendulum with parameters specified in the code stencil. The numerical integrator will advance the state (angle and velocity) of the pendulum in time given the current acceleration (generated from the pendulum equation of motion). If implemented successfully, this ideal pendulum should oscillate about the vertical (where the angle is zero) and with an amplitude that preserves the initial height of the pendulum bob.

Students enrolled in the undergraduate section will implement numerical integrators for:

- [Euler's Method](#)
- [Velocity Verlet](#)

For motion control, students in both undergraduate and sections will implement a [proportional-integral-derivative controller](#) to control the system's motor to a desired angle. This PID controller should output control forces integrated into the system's dynamics. You will need to tune the gains of the PID controller for stable and timely

motion to the desired angle for pendulum with parameters: length=2.0, mass=2.0, gravity=9.81.

For user input, you should be able to:

- select the choice of integrator using the [0-4] keys (with the "none" integrator as a default),
- toggle the invocation of the servo controller with the 'c' or 'x' key (which is off by default),
- decrement and increment the desired angle of the 1 DOF servoed robot arm using the 'q' and 'e' keys, and
- (for the double pendulum) decrement and increment the desired angle of the second joint of the arm using the 'w' and 'r' keys, and
- momentarily disable the servo controller with 's' key (and allowing the arm to swing uncontrolled).

## Graduate Section Requirement

Students enrolled in the graduate section will implement numerical integrators for:

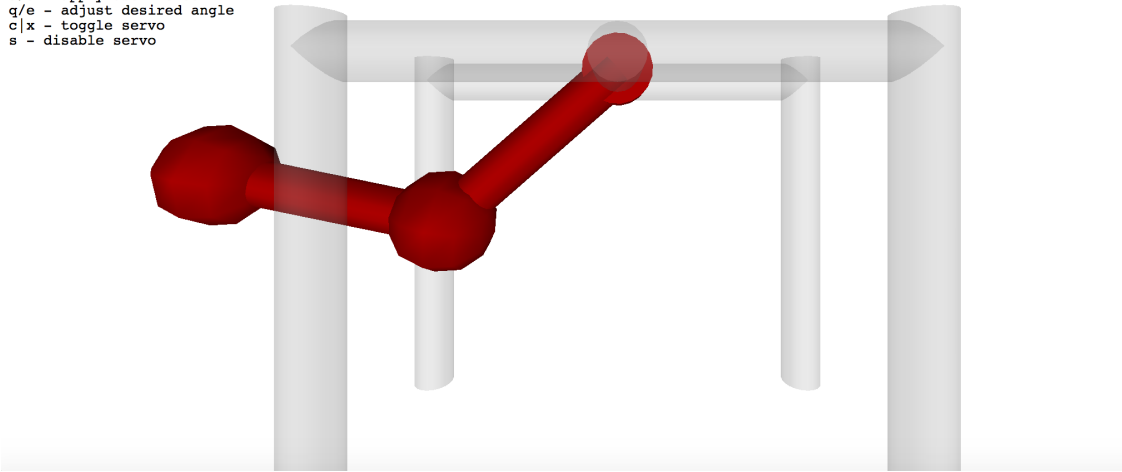
- [Euler's Method](#)
- [Verlet integration](#)
- [Velocity Verlet](#)
- [Runge-Kutta 4](#)

to simulate and control both a single pendulum (in "pendulum1.html") and a double pendulum (by creating "pendulum2.html"). A code stencil update will be provided for pendulum2. The double pendulum is allowed to have a smaller timestep than the single pendulum, within reasonable limits. A working visualization for the double pendulum will look similar to the following:

```
System
t = 55.50 dt = 0.05
integrator = verlet
x = -0.85 -0.92
x_dot = 0.97 -0.44
```

```
Pendulum
mass = 2.00 2.00
length = 2.00 2.00
gravity = 9.81
```

```
Keys
[0-4] - select integrator
a/d - apply user force
q/e - adjust desired angle
c|x - toggle servo
s - disable servo
```



## ADVANCED EXTENSIONS

Of the 4 possible advanced extension points, one additional point for this assignment can be earned by generating a random desired setpoint state and using PID control to your Pendularm to this setpoint. This code must randomly generate a new desired setpoint and resume PID control once the current setpoint is achieved. **A setpoint is considered achieved if the current state matches the desired state upto 0.01 radians for 2 seconds.** The number of setpoints that can be achieved in 60 seconds must be maintained and reported in the user interface. The invocation of this setpoint trial must be enabled a user pressing the "t" key in the user interface.

Of the 4 possible advanced extension points, two additional points for this assignment can be earned by implementing a simulation of a planar [cart pole system](#). This cartpole system should have joint limits on its prismatic joint and no motor forces applied to the rotational joint. This cart pole implementation should be contained within the file "cartpole.html" under the "project\_pendularm" directory.

Of the 4 possible advanced extension points, two additional points for this assignment can be earned by implementing a single pendulum simulator in maximal coordinates with a spring constraint enforced by [Gauss-Seidel optimization](#). This maximal coordinate pendulum implementation should be contained within the file "pendularm1\_maximal.html" under the "project\_pendularm" directory. An additional

point can be earned by extending this implementation to a cloth simulator in the file "cloth\_pointmass.html".

## Project Submission

For turning in your assignment, push your updated code to the **master** branch in your repository.

## Assignment 3: Forward Kinematics

**Due 11:59pm, Wednesday, October 16, 2019**

Forward kinematics (FK) forms the core of our ability to purposefully control the motion of a robot arm. FK will provide us general formulation for controlling any robot arm to reach a desired configuration, and execute a desired trajectory. Specifically, FK allows us to predict the spatial layout of the robot in our 3D world given a configuration of its joints. For the purposes of grasping and dexterous tasks, FK gives us the critical ability to predict the location of the robot's hand (i.e., endeffector). As shown in our [IROS 2017](#) video below, such manipulation assumes a robot has already perceived its environment as scene estimate of objects and their position and orientation. Given this scene estimate, a robot controller uses FK to evaluate and execute viable endeffector trajectories for grasping and manipulating an object.

**SUM: Sequential Scene Understanding and Manipulation**



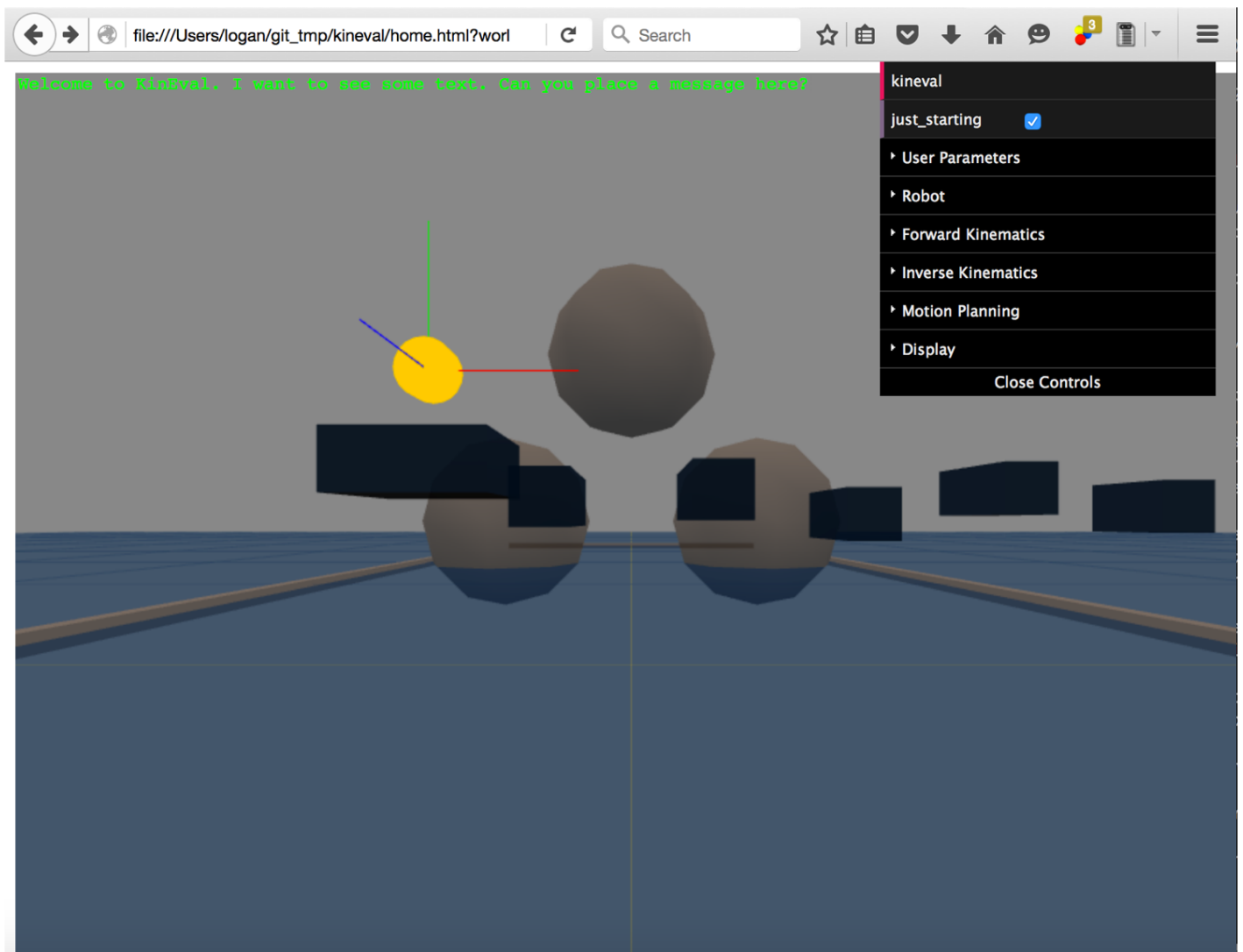
In this assignment, you will render the forward kinematics of an arbitrary robot, given an arbitrary kinematic specification. A collection of specifications for various robots is provided in the "robots" subdirectory of the KinEval code stencil. These robots include the Rethink Robotics' Baxter and Sawyer robots, the Fetch mobile manipulator, and a variety of example test robots, as shown below. To render the robot properly, you will compute matrix coordinate frame transforms for each link and joint of the robot based on the parameters of its hierarchy configuration. The computation of the matrix transform for each joint and link will allow KinEval's rendering support routines to properly display the full robot. We will assume the joints will remain in their zero position, saving joint motion for the next assignment.

### **Just Starting Mode**

`kineval_stencil` contains a code template for this assignment as well as all future projects in the course. This KinEval stencil allows for developing the core of a modeling and control computation stack (forward kinematics, inverse kinematics, and motion planning) in a modular fashion.

If you open "home.html" in this repository, you should see the disconnected pieces of a robot bouncing up and down, similar to the snapshot below for "robots/mr2.js"). This initial mode is the "starting point" state of the stencil to help build familiarity with JavaScript/HTML5 and KinEval.





Your task is to make these objects in starting point mode responsive to keyboard commands. Specifically, these objects will move upward, stop/start jittering, move closer together, and further apart (although more is encouraged). To do this, you will modify "kineval/kineval\_startingpoint.js" at the sections marked with "STENCIL". These sections also include code examples meant to be a quick (and very rough) introduction to JavaScript and homogeneous transforms for translation, assuming programming competency in another language.

## Brief KinEval Stencil Overview

Within the KinEval stencil, the functions `my_animate()` and `my_init()` in "home.html" are the principal access points into animation system. `my_animate()` is particularly important as it will direct the invocation of functions we develop throughout the AutoRob course. `my_animate()` and `my_init()` are called by the primary process that maintains the animation loop: `kineval.animate()` and `kineval.init()` within

"kineval/kineval.js".

**IMPORTANT:** "kineval/kineval.js", `kineval.animate()`, `kineval.init()`, and any of the given robot descriptions should not be modified.

For starting point mode, `my_animate()` will call `startingPlaceholderAnimate()` and `startingPlaceholderInit()`. `startingPlaceholderInit()` contains JavaScript tutorial-by-example code that initializes variables for this project. `startingPlaceholderAnimate()` contains keyboard handlers and code to update the positioning of each body of the robot. By modifying the proper variables at the locations labeled "STENCIL", this code will update the transformation matrix for each geometry of the robot (stored in the ".xform" attribute) as a translation in the robot's world. The ".xform" transform for each robot geometry is then used by `kineval.robotDraw()` to have the browser render the robot parts in the appropriate locations.

## Forward Kinematics

Assuming proper completion of Just Starting Mode, you are now ready for implementation of robot forward kinematics. Ensure the following files are included (within script tags) in your "home.html". You will modify these files for implementing FK:

- "kineval/kineval\_robot\_init.js" for initializing your robot object based on a given description object; modification is required to add parent and child references to each link
- "kineval/kineval\_forward\_kinematics.js" for implementing (a recursive) traversal over joints and links to compute transforms; traversal of forward kinematics is invoked from `kineval.robotForwardKinematics()` within `my_animate()` in home.html
- "kineval/kineval\_matrix.js" for the implementation of your vector and matrix routines, such as for matrix multiplication, matrix generation, etc. in `kineval_matrix.js`

## Robot Examples and Initialization

Each file in the "robots" subdirectory contains code to create a robot data object . This data object is initialized the kinematic description of a robot (as well as some meta information and rendering geometries). The kinematic description defines a hierarchical configuration of the robot's links and joints. This description is a subset of the [Unified Robot Description Format \(URDF\)](#) converted into JSON format. The basic features of URDF are described in [this tutorial](#).

**IMPORTANT (seriously):** The given robot description files should **NOT** be modified. Code that requires modified robot description files will fail tests used for grading. You are welcomed and encouraged to create new robot description files for additional testing.

The selection of file with a robot description can occur directly in the URL for "home.html". As a default, the "home.html" in the KinEval stencil assumes the "mr2" robot description in "robots/robot\_mr2.js". Another robot description file can be selected directly in the URL by adding a robot parameter. This parameter is segmented by a question mark and sets the robot file pointer to a given file local location, relative to "home.html". For example, a URL with "home.html?robot=robots/robot\_urdf\_example.js" will use the URDF example description.

In addition to the given initialization, you should extend the robot object to complete the kinematic hierarchy to specify the parent and children of each link. This modification should be made in the `kineval.initRobotJoints()` function in "kineval/kineval\_robot\_init.js". The children array of a link should always be defined, which would result in an empty array for leaf nodes in the kinematic tree. For the KinEval user controls to work properly, the children array should be named as the ".children" property of the link.

**Note:** KinEval refers to links and joints as strings, not pointers, within the robot object. `robot.joints` (as well as `robot.links`) is an array of data objects that are indexed by strings. Each of these objects stores relevant fields of information about the joint, such as its transform (".xform"), parent (".parent") and child (".child") in the kinematic hierarchy, local transform information (".origin"), etc. As such, `robot.joints['JointX']` refers to an object for a joint. In contrast, `robot.joints['JointX'].child` refers to a string ('LinkX'), that can then be used to reference a link object (as `robot.links['LinkX']`).

Similarly, `robot.links['LinkX'].parent` refers to a joint as a string 'JointX' that can then be used to reference a joint object in the `robot.joints` array.

## Invoking Forward Kinematics

The function `kineval.robotForwardKinematics()` in "`kineval/kineval_forward_kinematics.js`" will be the main point of invocation for your FK implementation. This function is responsible for updating matrix transforms for the frame of each link and joint with respect to the global world coordinates. The computed transform for each frame of the robot needs to be stored in the `".xform"` field. For a given link named 'LinkX', this `xform` field can be accessed as `robot.links['LinkX'].xform`. For a given joint named 'JointX', this `xform` field can be accessed as `robot.joints['JointX'].xform`. Once `kineval.robotForwardKinematics()` completes, the updated transforms for each frame are used by the function `kineval.robotDraw()` in the support code to render the robot.

A matrix stack recursion can be used to compute these global frames, starting from the base of the robot (specified as a string in `robot.base`). This recursion should use local translation and rotation parameters of each joint in relation to its parent link in its traversal of the hierarchy. For a given joint 'JointX', these translation and rotation parameters are stored in the robot object as `robot.joints['JointX'].origin.xyz` and `robot.joints['JointX'].origin.rpy`, respectively. The current global translation and rotation for the base of the robot (`robot.base`) in the world coordinate frame is stored in `robot.origin.xyz` and `robot.origin.rpy`, respectively.

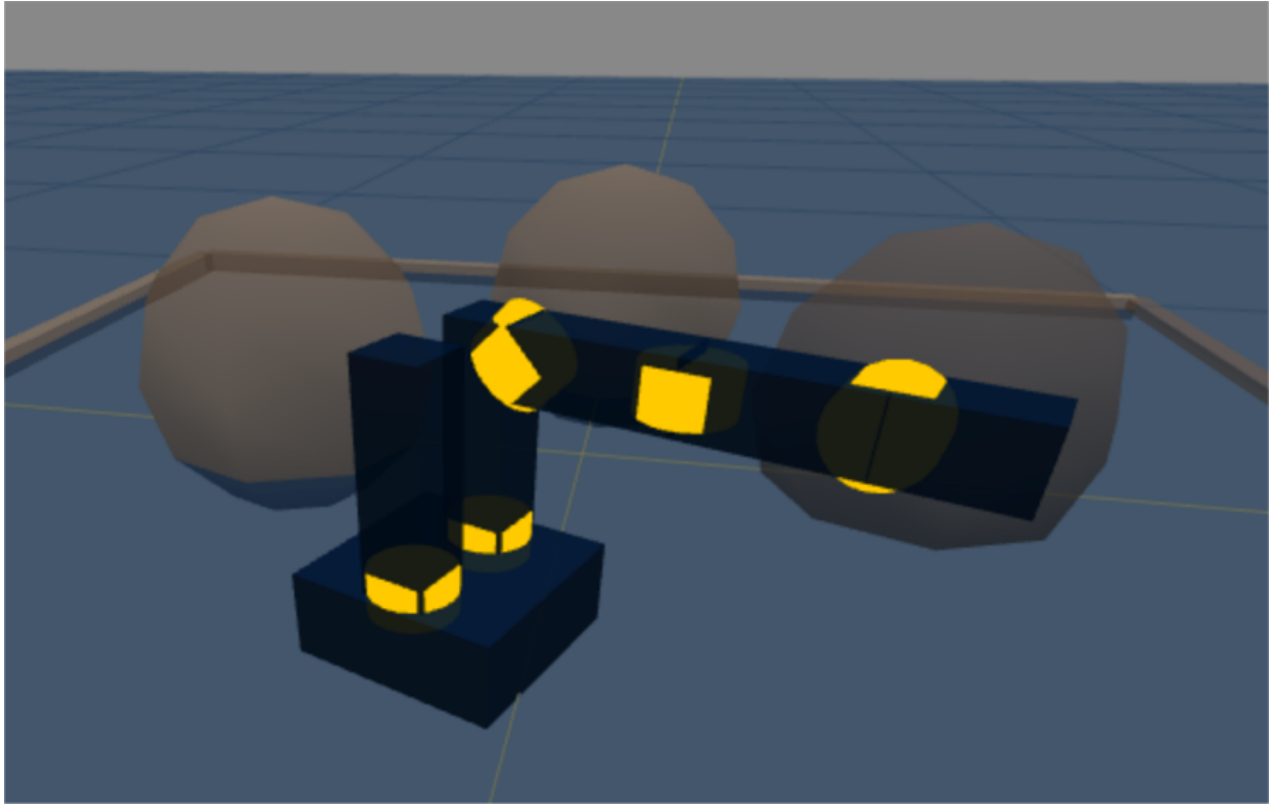
To run your FK routine, you must toggle out of starting point mode. This toggle can be done interactively within the GUI menu or by setting `kineval.params.just_starting` to `false`. The code below in "`home.html`" controls starting point mode invocation, where single line can be uncommented to use FK mode by default:

```
// set to starting point mode is true as default
// set to false once starting forward kinematics project
//kineval.params.just_starting = true;

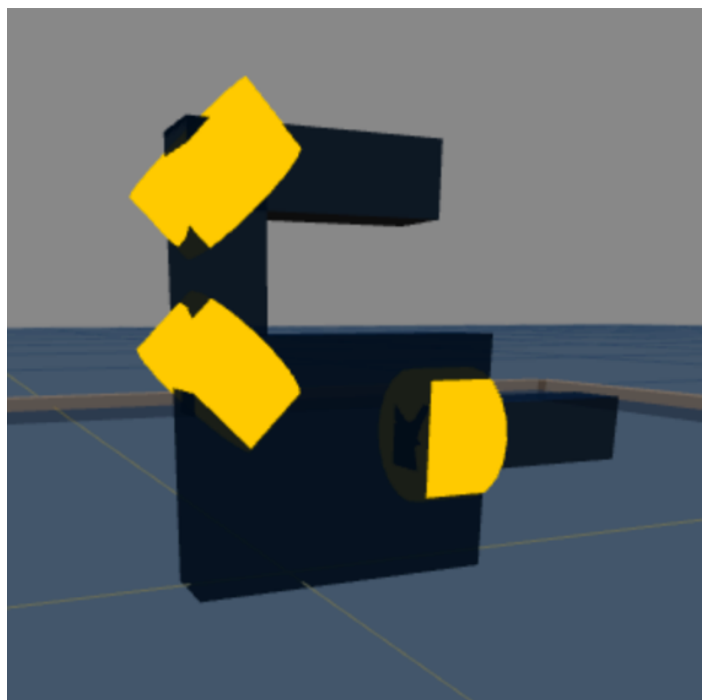
if (kineval.params.just_starting == true) {
    startingPlaceholderAnimate();
    kineval.robotDraw();
}
```

```
    return;  
}
```

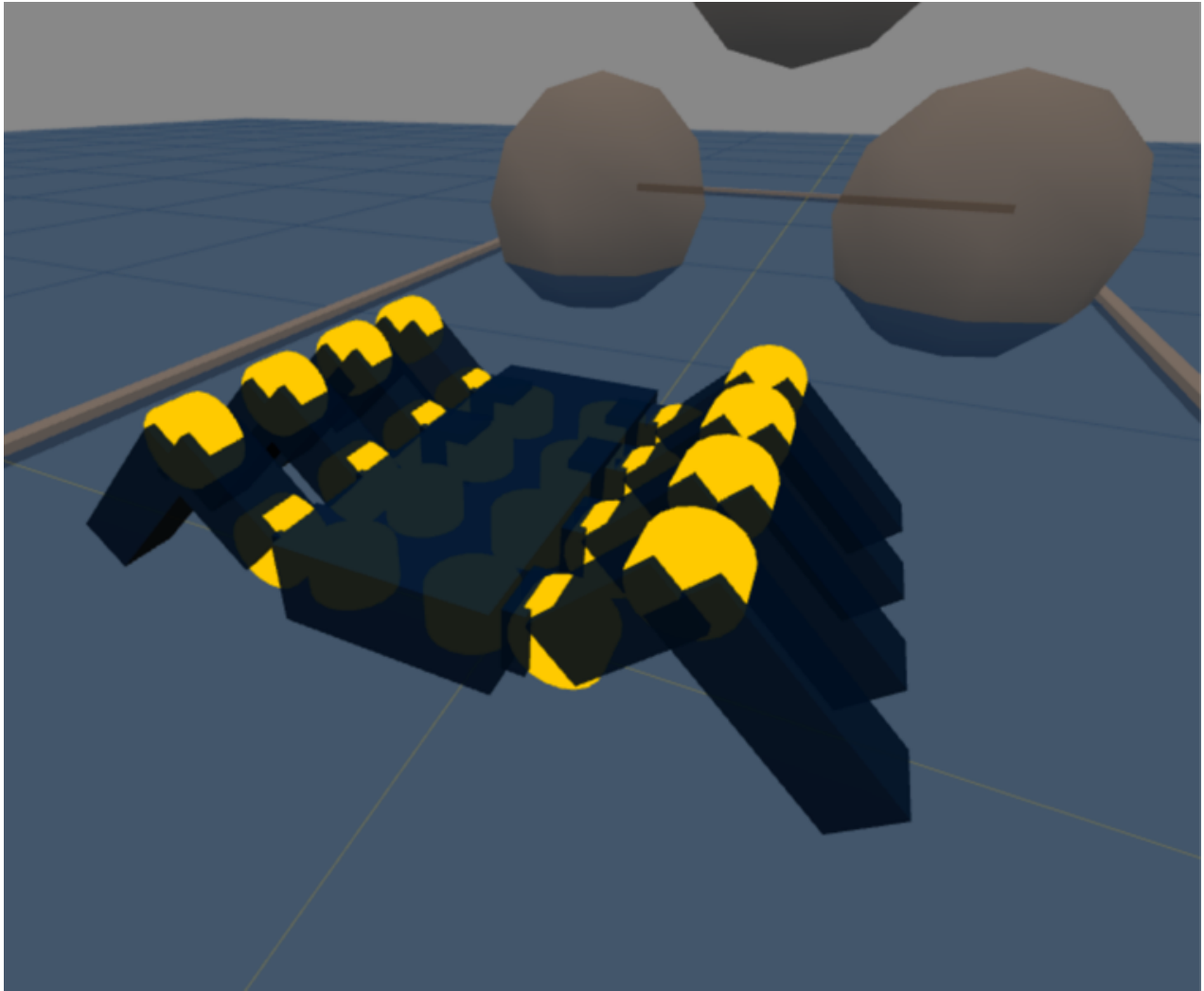
The "robots/robot\_mr2.js" example should produce the following:



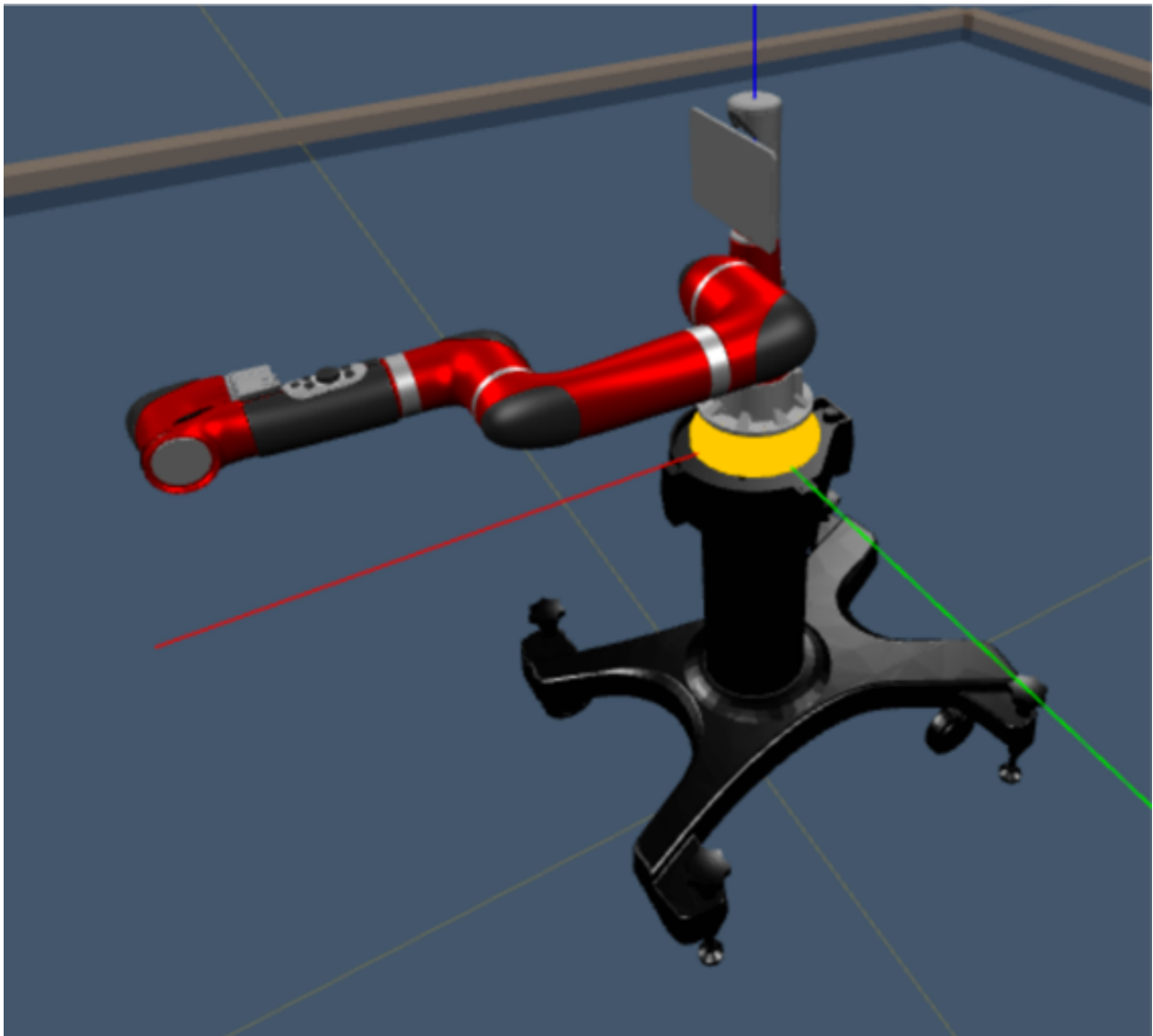
If implemented properly, the "robots/robot\_urdf\_example.js" example should produce the following rendering:



The "robots/robot\_crawler.js" example should produce the following (shown with joint axes highlighted):



The "robots/sawyer/sawyer.urdf.js" example should produce the following (shown with joint axes highlighted):



## Interactive Hierarchy Traversal

Additionally, a correct implementation will be able to interactively traverse the kinematic hierarchically by changing the active joint. The active joint has focus for user control, which will be used in the next assignment. For now, we are using the active joint to ensure your kinematic hierarchy is correct. You should be able to move up and down the kinematic hierarchy with the "k" and "j" keys, respectively. You can also move between the children of a link using the "h" and "l" keys.

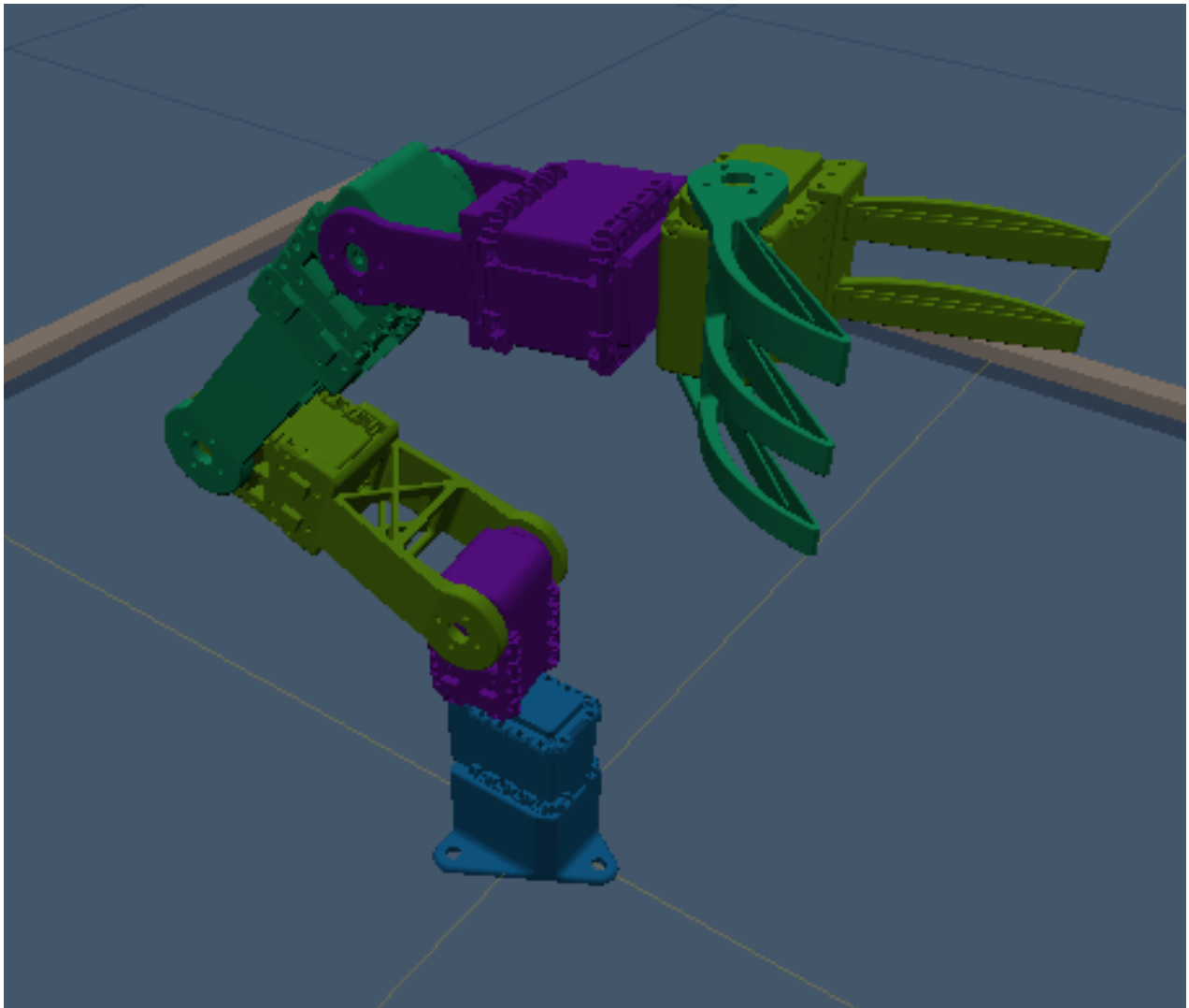
## Orienting Joint Rendering Cylinders

The cylinders used as rendering geometries for joints are not aligned with joint axes by default. The support code in Kineval will properly orient joint rendering cylinders. To use this functionality, simply implement a vector cross product function named

`vector_cross()` in your matrix routines. `vector_cross()` will be automatically detected and used to properly orient each joint rendering cylinder.

### Undergraduate Advanced Extension

Students in the AutoRob Undergraduate Section can earn one additional point by creating a robot description for the RexArm 4-DOF robot arm, which can be used later in [EECS 467 \(Autonomous Robotics Laboratory\)](#). [Rexarm link geometries](#) are provided in [STL format](#). [RoBob Ross](#) is an example of a RexArm project from 467 in Winter 2017. Below is a snapshot of a RexArm in KinEval created by mattdr:



### Graduate Section Requirement

Students in the AutoRob Graduate Section must: 1) implement the assignment as described above to work with given examples, which includes the Fetch, Baxter, and

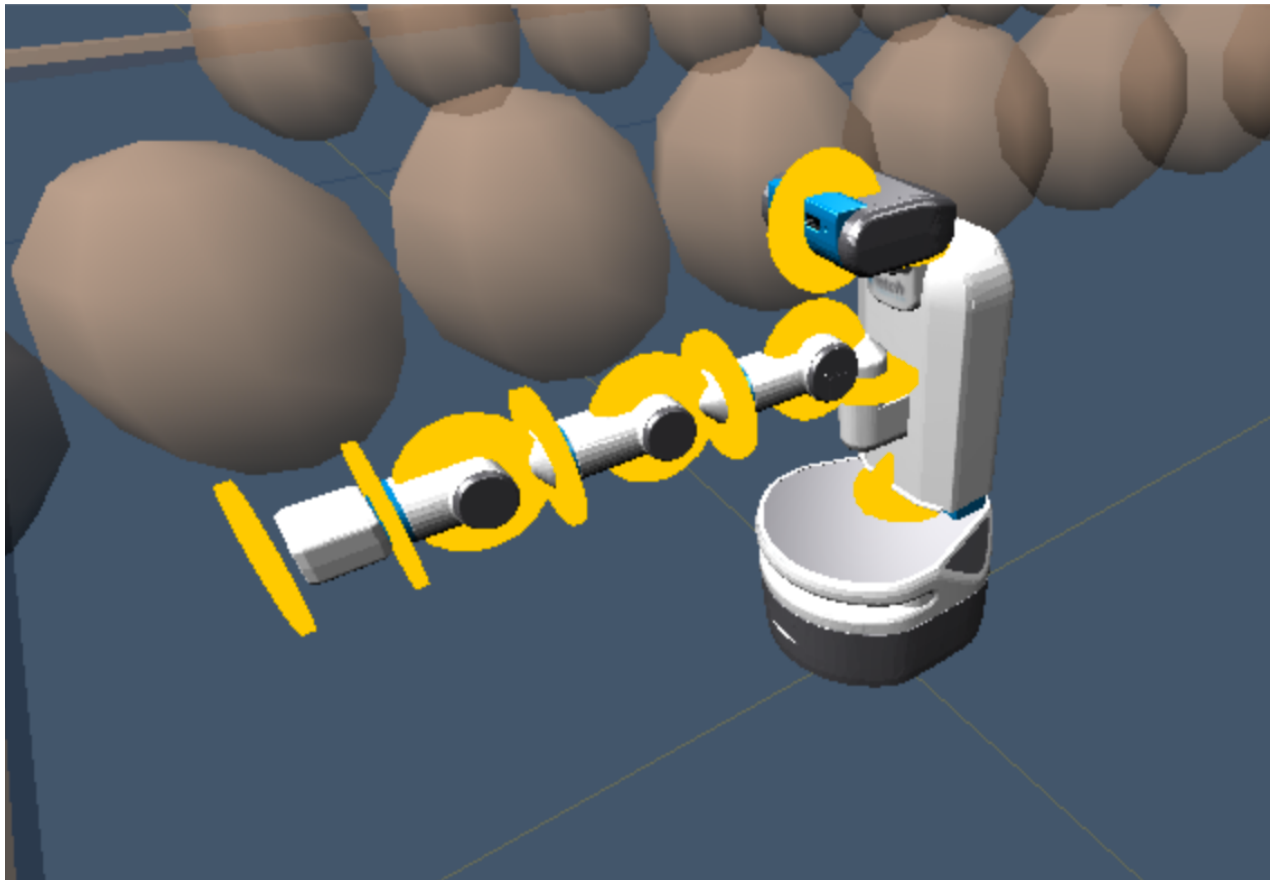


Sawyer robot descriptions, and 2) create a new robot description that works with KinEval.

The files "robots/fetch/fetch.urdf.js" and "robots/baxter/baxter.urdf.js" contain the robot data object for the Fetch kinematic description. The Fetch robot JavaScript file is converted from the [Fetch URDF description](#) for ROS. A similar process was also done for the [Baxter URDF description](#).

ROS uses a different default coordinate system than threejs, which needs to be taken into account in the FK computation. Coordinate frames in ROS assumes that the Z, X, and Y axes correspond to the up, forward, and side directions, respectively. In contrast, threejs coordinate frames assume the Y, Z, and X correspond to the up, forward, and side directions. The variable `robot.links_geom_imported` will be set to true when geometries have been imported from ROS and set to false when geometries are defined completely within the robot description file.

A proper implementation for `fetch.urdf.js` description should produce the following (shown with joint axes highlighted):



The newly created robot description should be placed in the "robots" directory with a filename with your username in the format "robot\_uniqueid.js" if no external geometries are used for this robot (similar to the MR2 or Crawler robots). If external geometries are imported (similar to the Fetch and Baxter), the robot description should be in a new subdirectory with the robot's name. The robot's name should also be used to name the URDF file, such as "robots/newrobotname/newrobotname.urdf.js". It is requested that geometries for a new robot go into this directory as a "meshes" subdirectory, such as "robots/newrobotname/meshes". Guidance can be provided during office hours about creating or converting URDF-based robot description files to KinEval-compliant JavaScript and importing Collada, STL, and Wavefront OBJ geometry files.

Students are highly encouraged to port URDF descriptions of real world robot platforms into their code. Such examples of real world robot systems include the [Kinova Movo](#), [NASA Valkyrie](#) and [Robonaut 2](#), [Boston Dynamics Atlas](#), [Universal Robots UR10](#), and [Willow Garage PR2](#).

The following KinEval-compatible robot descriptions were created by students in past offerings of the AutoRob course. These descriptions are available for your use:

- Boston Dynamics [Atlas](#) by yeyangf
- Agility Robotics [Cassie](#) by mungam
- NASA [Robonaut 2](#) by nikhita
- Human Support Robot by sajanptl
- KUKA [Lightweight Arm](#) by nmtvijay
- [R2D2-like robot](#) by eeyan
- Universal Robots [UR10](#) by chengyah
- [Wall-E-like robot](#) by sarahcc

ADVANCED EXTENSIONS

Of the 4 possible advanced extension points, two additional points for this assignment can be earned by generate a proper Denavit-Hartenberg table for the kinematics of the Fetch robot. This table should be placed in the "robots/fetch" directory in the file "fetchDH.txt".

Of the 4 possible advanced extension points, three additional points for this assignment can be earned by implementing LU decomposition (with pivoting) routines for matrix inversion and solving linear systems. These functions should be named "matrix\_inverse" and "linear\_solve" and placed within the file containing your matrix routines.

Of the 4 possible advanced extension points, three additional points for this assignment can be earned by implementing rigid body transformations as dual quaternions (Kenwright 2012), in addition to the products of exponentials method described in class. Use of dual quaternion transformations must be selectable from the KinEval user interface.

## **Project Submission**

For turning in your assignment, push your updated code to the **master** branch in your repository.

## **Assignment 4: Robot FSM Dance Contest**

**Due 11:59pm, Wednesday, October 30, 2019**

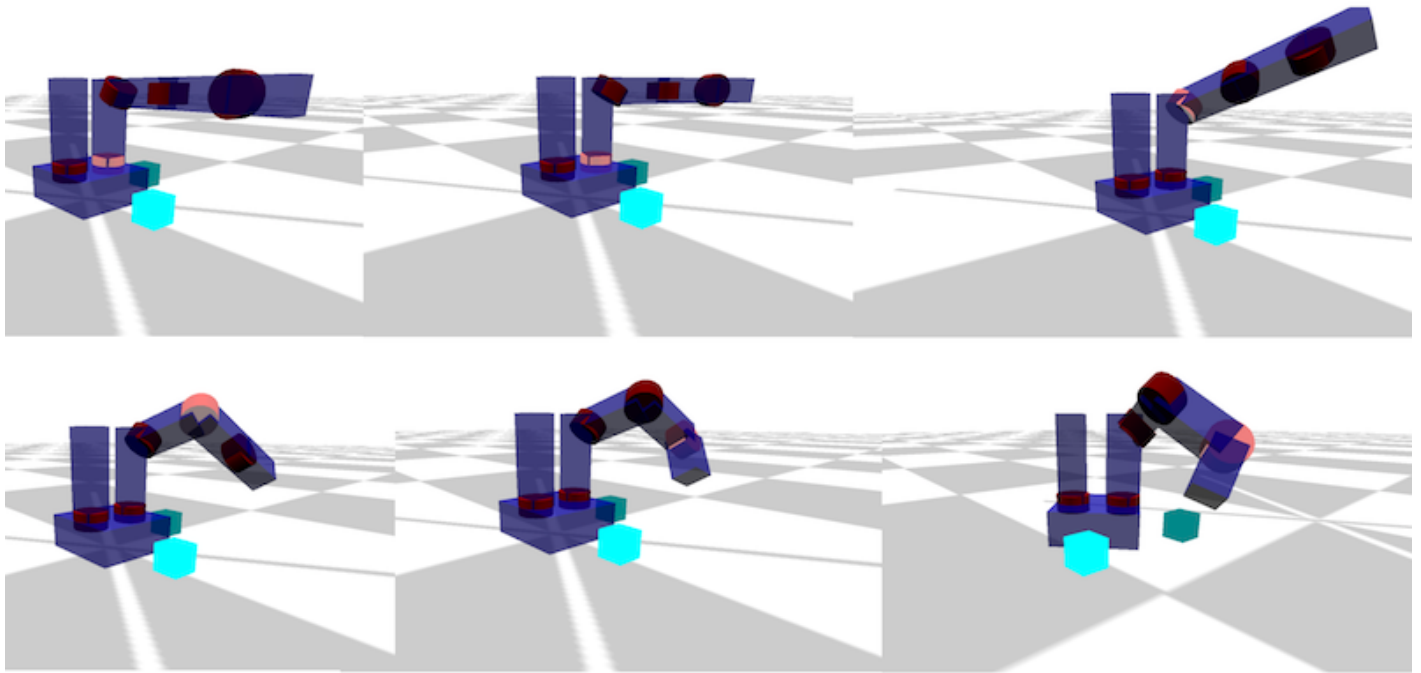
Executing choreographed motion is the most common use of current robots. Robot choreography is predominantly expressed as a sequence of setpoints (or desired states) for the robot to achieve in its motion execution. This form of robot control can be found among a variety of scenarios, such as robot dancing (video below), GPS navigation of autonomous drones, and automated manufacturing. General to these robot choreography scenarios is a given setpoint controller (such as our PID controller from Pendularm) and a sequence controller (which we will now create).

## NAO Robots Thriller Dance

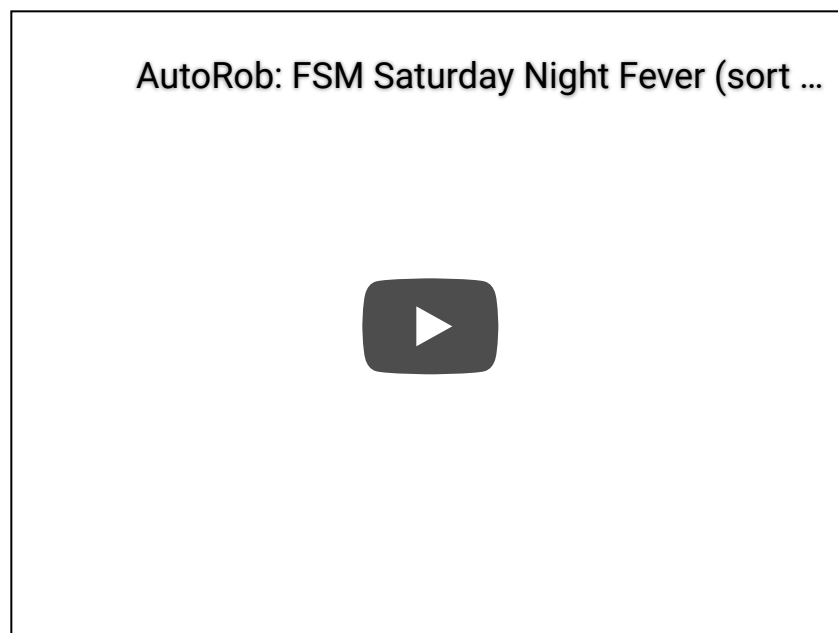


For this assignment, you will build your own robot choreography system. This choreography system enable a robot to execute a dance routine by adding motor rotation to its joints and creating a Finite State Machine (FSM) controller over pose setpoints. Your FK implementation will be extended to consider angular rotation about each joint axis using quaternions for axis-angle rotation. The positioning of each joint with respect to a given pose setpoint will be controlled by an simple P servo implementation (based on the Pendularm assignment). You will implement an FSM controller to update the current pose setpoint based on the robot's current state and predetermined sequence of setpoints. For a single robot, you will choreograph a dance for the robot by creating an FSM with your design of pose setpoints and an execution sequence.

This controller for the "mr2" example robot was a poor attempt at [robot Saturday Night Fever](#) (please do better):



This [updated dance](#) controller for the Fetch robot is a bit better, but still very far from optimal:



Assuming proper completion of Assignment 4, ensure the following files are included (within script tags) in your "home.html". You will modify these files for implementing axis-angle rotation, the pose setpoint controller, and the FSM controller:

- "kineval/kineval\_quaternion.js" for your implementation of quaternions for axis-angle rotation in 3D

- "kineval/kineval\_forward\_kinematics.js" to augment your existing kinematic traversal to account for axis-angle joint rotation
- "kineval/kineval\_controls.js" includes function `kineval.applyControls()` to apply a control update to the robot's base and angle of each joint, as well as updating the camera position; this update just does an addition and does not consider a physical model of dynamics
- "kineval/kineval\_servo\_control.js" for your implementation of a P servo controller and an FSM pose sequencer

## Joint Axis Rotation and Interactive Control

Each joint of the robot needs several additional properties for joint rotation and control. These joint properties for the current angle rotation (".angle"), applied control (".control"), and servo parameters (".servo") have already been created within the function `kineval.initRobotJoints()`. The joint's angle will be used to calculate a rotation about the joints (normal) axis of rotation vector, specified in the ".axis" field. The 3D rotation due to joint movement should be accounted for in the robot's forward kinematics and implemented as quaternions in "kineval/kineval\_quaternion.js".

If joint axis rotation is implemented correctly, you should be able to use the 'u' and 'i' keys to move the currently active joint. These keys respectively decrement and increment the ".control" field of the active joint. Through the function `kineval.applyControls()`, this control value effectively adds an angular displacement to the joint angle.

## Interactive Base Movement Controls

The user interfaces also enables controlling the global position and orientation of the robot base. In addition to joint updates, the system update function `kineval.applyControls()` also updates the base state (in `robot.origin`) with respect to its controls (specified in `robot.controls`). With the support function `kineval.handleUserInput()`, the 'wasd' keys are purposed to move the robot on the ground plane with 'q' and 'e' keys for lateral base movement. In order for these keys to behave properly, the heading and lateral directions of the robot base are needed such

that they respectively express coordinates along local z-axis and x-axis of the base in the global frame. These vectors need to be computed within your FK implementation and stored within two global variables: `robot_heading` and `robot_lateral`. Each of these variables should be a homogeneous 3D vector stored as a 2D array.

If `robot_heading` and `robot_lateral` are implemented properly, the robot should now be interactively controllable in the ground plane.

## Pose Setpoint Controller

Once joint axis rotation is implemented, you will implement proportional setpoint controller for the robot joints in function `kineval.robotArmControllerSetpoint()` within `"kineval/kineval_servo_controller.js"`. This setpoint controller uses the current angle (`".angle"`), desired angle, and servo gains (specified in the `".servo"` object) of each joint to output a control (`".control"`) for the joint. The desired angle for a joint 'JointX' is stored in `kineval.params.setpoint_target['JointX']` as a scalar. All of these joint object properties are initialized in the function `kineval.initRobotJoints()` in `"kineval/kineval_robot_init.js"`.

For testing, a "clock movement" controller has been provided as the function `setpointClockMovement()` in `"kineval/kineval_setpoint_controller.js"`. This function can be invoked by holding down the 'c' key or from the UI. This controller goes well with [this song](#).

The robot can servo to the current pose setpoint by holding down the 'o' key or selecting 'persist\_pd' from the UI. Pressing the '0' key sets the current setpoint to the zero pose, where all joint angles are zero. Stored in `kineval.setpoints`, up to 9 other arbitrary pose setpoints can be stored by KinEval for pose control. The current robot pose can be interactively stored by pressing "Shift+number\_key" (e.g., "Shift+1"). The current setpoint can be assigned a stored pose by pressing one of the non-zero number keys [1-9]. At any time, the currently stored setpoints can be output to the console as JavaScript code using the `JSON.stringify` function for the setpoint object, as the statement `"JSON.stringify(kineval.setpoints);"`. This setpoint array can be included in your code as part of your dance controller.

## FSM Controller

Once your pose setpoint controller is work, a FSM controller should be implemented in the function `kineval.setpointDanceSequence()` in "kineval/kineval\_setpoint\_control.js". The reference implementation uses pose setpoints initialized and stored in `kineval.setpoints` with a sequence of indices stored in `kineval.params.dance_sequence_index` and playback index stored in `kineval.params.dance_pose_index`. If this convention is not used, the following line in "kineval/kineval\_userinput.js" will require modification:

```
if (kineval.params.update_pd_dance)
    textbar.innerHTML += "executing dance routine, pose " +
kineval.params.dance_pose_index + " of " +
kineval.params.dance_sequence_index.length;
```

## Graduate Section Requirement

Students in the graduate section of AutoRob must implement the assignment as described above for the Fetch and Baxter robots with two additional requirements: 1) proper enforcement of joint types and limits for the robot description, and 2) integration (via [rosbridge](#)) of their code with ROS or a [Gazebo simulation of the Fetch](#).

The URDF JS files for these robots, included in the provided code stencil, contains joints with with various types that correspond to different types of motion:

- continuous: rotation about the joint axis with no joint limits
- revolute: rotation about the joint axis with joint limits
- prismatic: translation along the joint axis with joint limits
- fixed: no motion of the joint

Joints are considered to be continuous as the default. Joints with undefined motion types must be treated as continuous joints.

Your code can interface with any robot (or simulated robot) running `rosbridge/ROS` using the function `kineval.rosbridge()` in "kineval/kineval\_rosbridge.js". This code requires that the `rosbridge_server` package is running in a ROS run-time environment and listening on a websocket port, such as for `ws://fetch7:9090`. If your FK



implementation is working properly, the model of your robot in the browser will update along with the motion of the robot based on the topic subscription and callback. This functionality works seamlessly between real and simulated robots. Although this will not be done for this class, to control the robot arm, a rosbridge publisher must be written to update the ROS topic `"/arm_controller/follow_joint_trajectory/goal"` with a message of type `"control_msgs/FollowJointTrajectoryActionGoal"`.

Machines running rosbridge, ROS, and Gazebo for the Fetch will be available during special sessions of the class. Students are encouraged to install and run the Fetch simulator on their own machines based on [this tutorial](#).

#### ADVANCED EXTENSIONS

Of the 4 possible advanced extension points, one additional point for this assignment can be earned by adding the capability of displaying laser scans from a real or simulated Fetch robot.

Of the 4 possible advanced extension points, four additional points for this assignment can be earned by adding the capability of displaying 3D point clouds from a real or simulated Fetch robot and computing surface normals about each point.

Of the 4 possible advanced extension points, four additional points for this assignment can be earned by implementing dynamical simulation through the recursive [Newton-Euler algorithm](#) (Spong Ch.7). This dynamical simulation update be implemented as function `kineval.updateDynamicNewtonEuler()` in the file `"kineval/kineval_controls.js"`. In `"home.html"`, the call to `kineval.updateDynamicNewtonEuler()` should replace the call purely kinematic update in `kineval.applyControls()`.

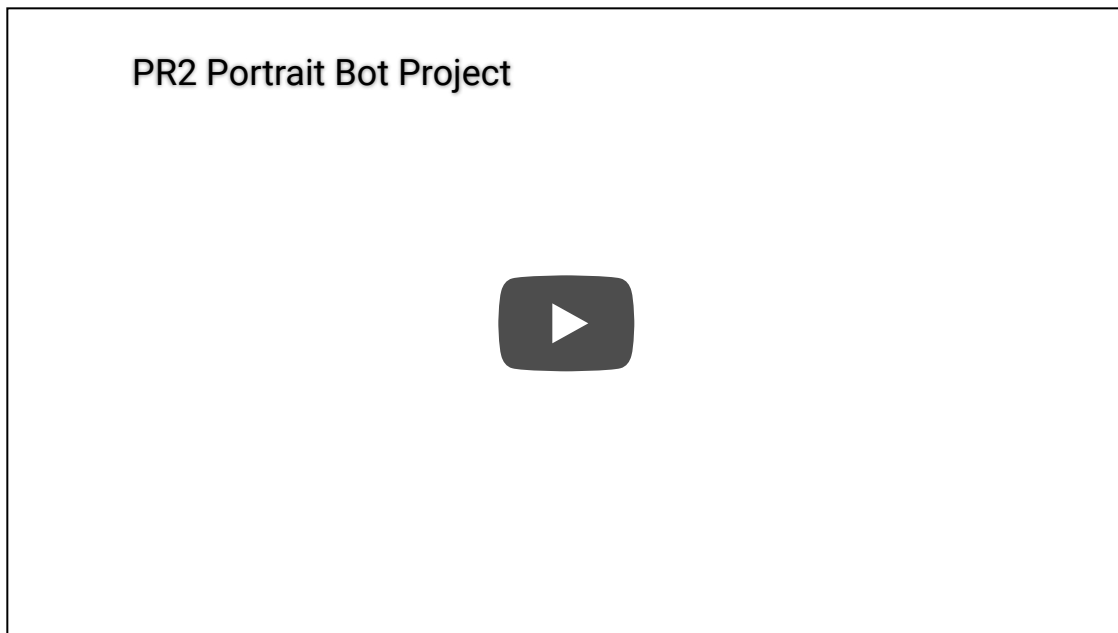
#### **Project Submission**

For turning in your assignment, push your updated code to the **master** branch in your repository.

## **Assignment 5: Inverse Kinematics**

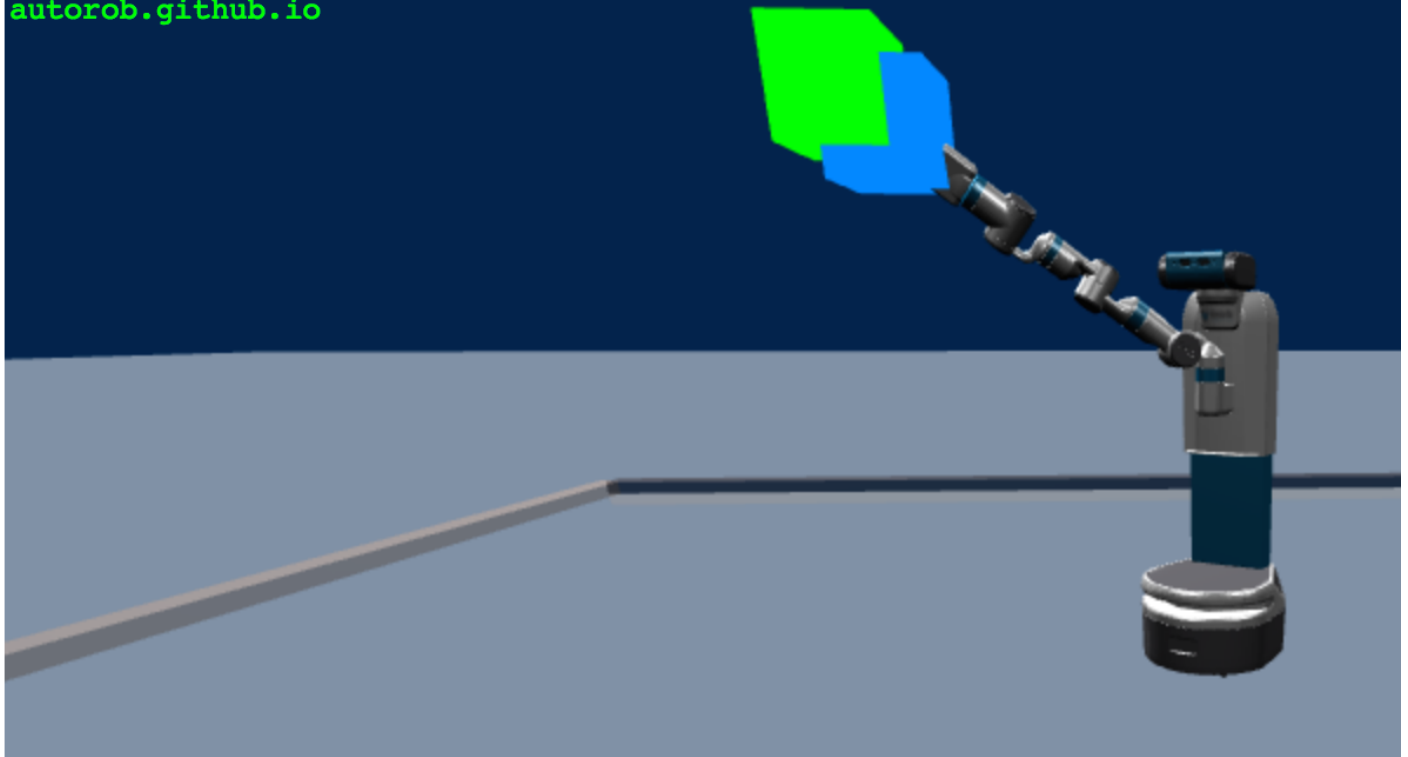
**Due 11:59pm, Friday, November 15, 2019**

Although effective, robot choreography in configuration space is super tedious and inefficient. This difficulty is primarily due to posing each joint of the robot at each setpoint. Further, changing one joint often requires updating several other joints due to the nature of kinematic dependencies. Inverse kinematics (IK) offers a much easier and efficient alternative. With IK implemented, we only need to pose the endeffector in a common workspace, and the states of the joints in configuration space automatically inferred. IK is also important when we are about the "tool tip" of an instrument being used by a robot. One such example is a robot using marker to draw a picture, such as in the PR2 Portrait Bot Project below:

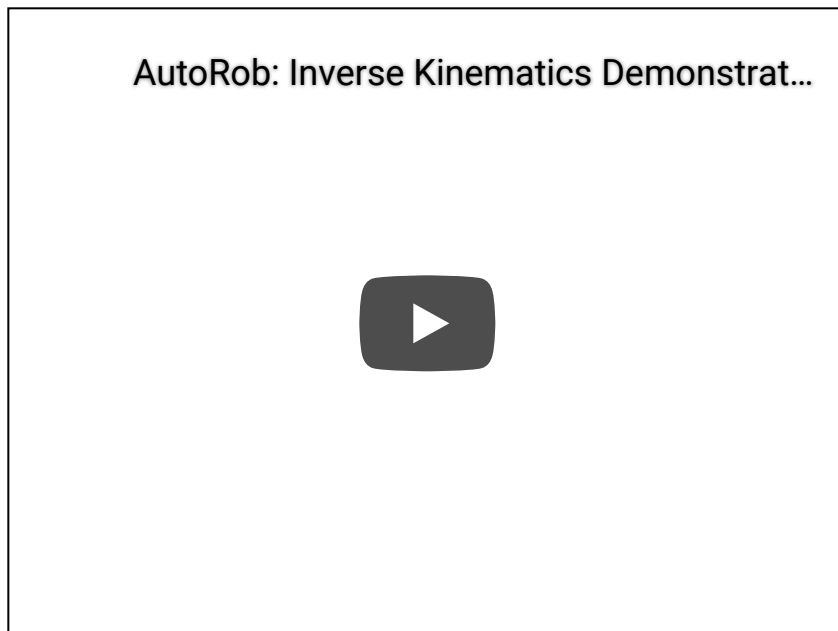


For this assignment, you will now control your robot to reach to a given point in space through inverse kinematics for position control of the robot endeffector. Inverse kinematics will be implemented through gradient descent optimization with both the Jacobian Transpose and Jacobian Pseudoinverse methods, although only one will be invoked at run-time.

autorob.github.io



As shown in the video below, if successful, your robot will be able to continually place its endeffector (indicated by the blue cube) exactly on the reachable target location (indicated by the green cube), regardless of the robot's specific configuration:



The core of this assignment is to complete the `kineval.iterateIK()` function in the file `kineval/kineval_inverse_kinematics.js`. This function is invoked within the function

`kineval.inverseKinematics()` with three arguments:

- `endeffector_target_world`: an object with two fields with the target endeffector position (as a 3D homogeneous vector) and orientation (as Euler angles) in the world frame
- `endeffector_joint`: the name of the joint directly connected to the endeffector
- `endeffector_position_local`: the location of the endeffector in the local joint frame

From these arguments and the current robot configuration, the `kineval.iterateIK()` function will compute controls for each joint. Upon update of the joints, these controls will move the configuration and endeffector of the robot closer to the target.

`kineval.iterateIK()` should also respect global parameters for using the Jacobian pseudoinverse (through boolean parameter `kineval.params.ik_pseudoinverse`) and step length of the IK iteration (through real-valued parameter `kineval.params.ik_steplength`). Kineval also maintains the current endeffector target information in the `kineval.params.ik_target` parameter.

IK iterations can be invoked through the user interface by holding down the 'p' key. Further, the 'r'/'f' keys will move the target location up/down. When performing IK iterations, the endeffector and its target pose will be rendered as cube geometries in blue and green, respectively.

In implementing this IK routine, please remember the following:

- Computation of the Jacobian need only to occur with respect to the joints along the chain from the endeffector joint to the robot base
- The location of the endeffector needs to be computed using transforms resulting from the robot's forward kinematics
- Matrix inversion can be invoked by using the provided routine `numeric.inv(mat)`, available through [numericjs](#)

- The computed velocity in configuration space should be applied to the robot through the `joint.controls` field of each joint

Students enrolled in EECS 398 will implement inverse kinematics for **only** the position of the endeffector.

## IK Random Trial

All students in the AutoRob course are expected to run their IK controller with the random trial feature in the KinEval stencil. The IK random trial is executed through the function `kineval.randomizeIKtrial()` in the file "`kineval/kineval_inverse_kinematics.js`". This function is incomplete in the provided stencil. Code for this function to properly run the random trial will be made available upon request through the course discussion channel.

## UNDERGRADUATE ADVANCED EXTENSION

Students in the AutoRob Undergraduate Section can earn one additional point by implementing a closed-form inverse kinematics solution for the RexArm 4-DOF robot arm, which can be used later projects in [EECS 467 \(Autonomous Robotics Laboratory\)](#).

## Graduate Section Requirement

Students enrolled in the graduate section of AutoRob will implement inverse kinematics for both the position and orientation of the endeffector, namely for the Fetch robot. The default IK behavior will be for endeffector position control. Both endeffector position and orientation are controlled when the boolean parameter `kineval.params.ik_orientation_included` is set to true.

## ADVANCED EXTENSIONS

Of the 4 possible advanced extension points, one additional point for this assignment can be earned by reaching to 100 targets in a random trial within 60 seconds. A video of this execution must be provided to demonstrate this achievement. This video file should be in the repository root directory with the name "IK100in60" and appropriate file extension.

Of the 4 possible advanced extension points, three additional points for this assignment can be earned by implementing the [Cyclic Coordinate Descent \(CCD\)](#) inverse kinematics algorithm by Wang and Chen (1991). This function should be implemented in the file "kineval/kineval\_inverse\_kinematics.js" as another option within the function `kineval.iterateIK()`.

Of the 4 possible advanced extension points, three additional points for this assignment can be earned by implementing [downhill simplex optimization](#) to perform inverse kinematics. This function should be implemented in the file "kineval/kineval\_inverse\_kinematics.js" as another option within the function `kineval.iterateIK()`.

Of the 4 possible advanced extension points, four additional points for this assignment can be earned by implementing resolved-rate inverse kinematics with null space constraints to respect joint limits. This function should be implemented in the file "kineval/kineval\_inverse\_kinematics.js" as another option within the function `kineval.iterateIK()`.

Of the 4 possible advanced extension points, four additional points for this assignment can be earned by extending your IK controller to use potential fields to avoid collisions.

## **Project Submission**

For turning in your assignment, ensure your completed project code has been committed and pushed to the *master* branch of your repository.

## **Assignment 6: Motion Planning**

**Due 11:59pm, Friday, December 6, 2019**

Our last programming project for AutoRob returns to search algorithms for generating navigation setpoints, but now for a high-dimensional robot arm. The A-star graph search algorithm in Project 1 is a good fit for path planning given the space to explore is limited to two degrees-of-freedom for a robot base. However, as the number of

degree-of-freedom of our robot increases, our search complexity will grow exponentially towards intractability. For such high-dimensional search problems, we now look to sampling-based search algorithms. These sampling-based algorithms trade off the guarantees and optimality of exhaustive graph search for viably tractable planning in complex environments. The example below shows one example of sample-based planning navigating to move a rod through a narrow passageway:



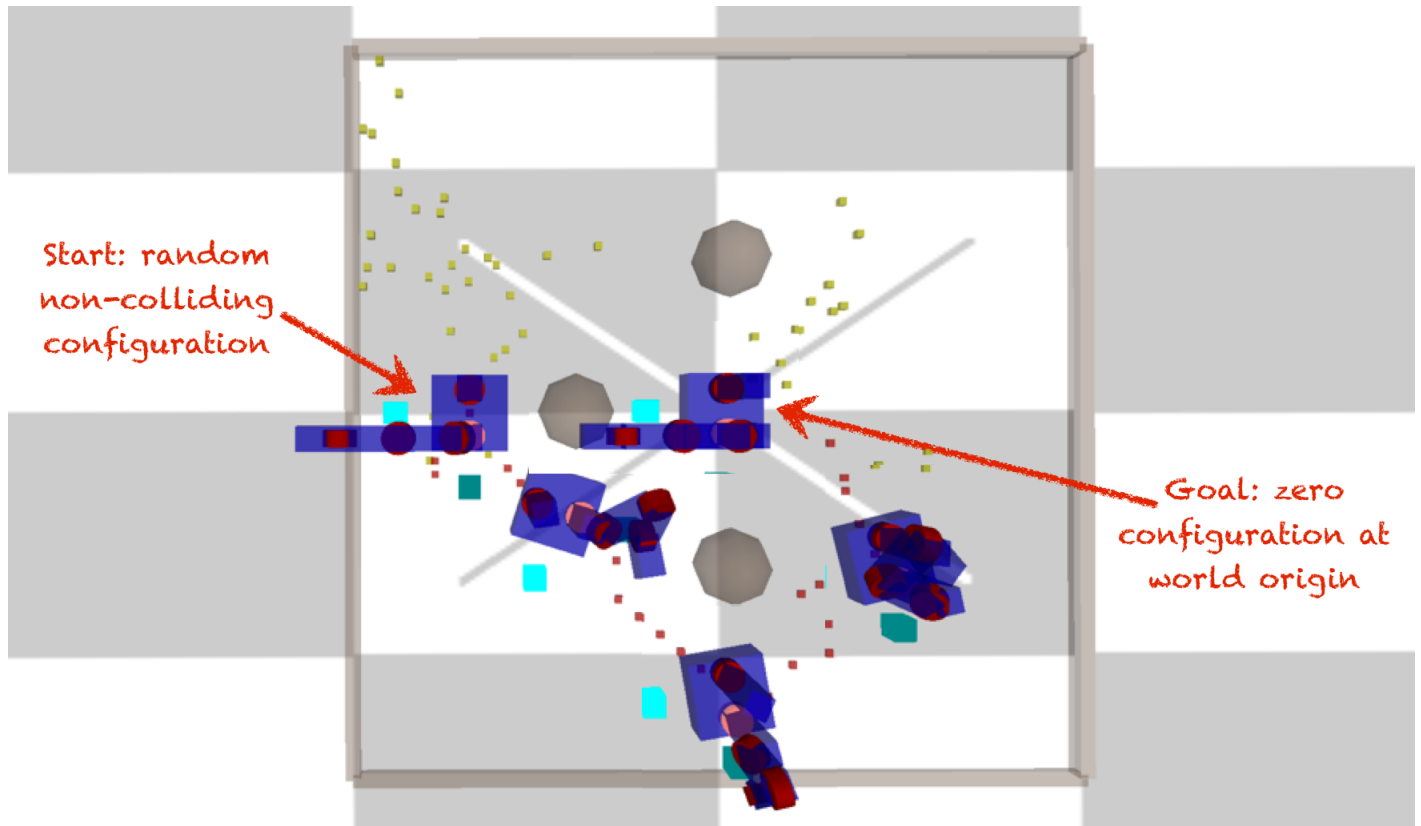
and such planning is also used in simple tabletop scenarios:



For this assignment, you will now implement a collision-free motion planner to enable your robot to navigate from a random configuration in the world to its home

configuration (or "zero configuration"). This home configuration is where every robot DOF has a zero value. For planning, configuration space includes the state of each joint and the global orientation and position of the robot base. Thus, the robot must move to its original state at the origin of the world. Motion planning will be implemented through the [RRT-Connect algorithm](#) (described by Kuffner and LaValle).

The graduate section will additionally implement the [RRT-Star](#) (alternate paper [link](#) via IEEE) motion planner of Karaman et al. (ICRA 2011).



The core of this assignment is to complete the `robot_rrt_planner_init()` and `robot_rrt_planner_iterate()` in the provided `kineval_rrt_connect.js` stencil. For successful execution, your implementation of RRT-Connect, the provided collision detection system, and a single specification of world geometry has been included in `home.html`:

```
< script src="kineval_rrt_connect.js" ></script>
< script src="kineval_collision.js" ></script>
< script src="worlds/world_basic.js" ></script>
```



The code stencil will automatically load a default world. A world can also be specified as an appended parameter within the URL, in the form of "?world=worlds/world\_name.js". The result of including a world file are the global objects "robot\_boundary" describing the min and max values of the world boundaries along the X, Y, and Z axes and "robot\_obstacles" as the locations and radii of sphere obstacles. To ensure these worlds rendered in the display and available for collision detection, the geometries of the world are included through the provided call to `kineval.initWorldPlanningScene()` in `kineval/kineval.js`.

Note: your planner should be constrained such that the search does not consider configurations where the base is outside the X-Z plane. Specifically, the base should not translate along the Y axis, and should not rotate about the X and Z axes.

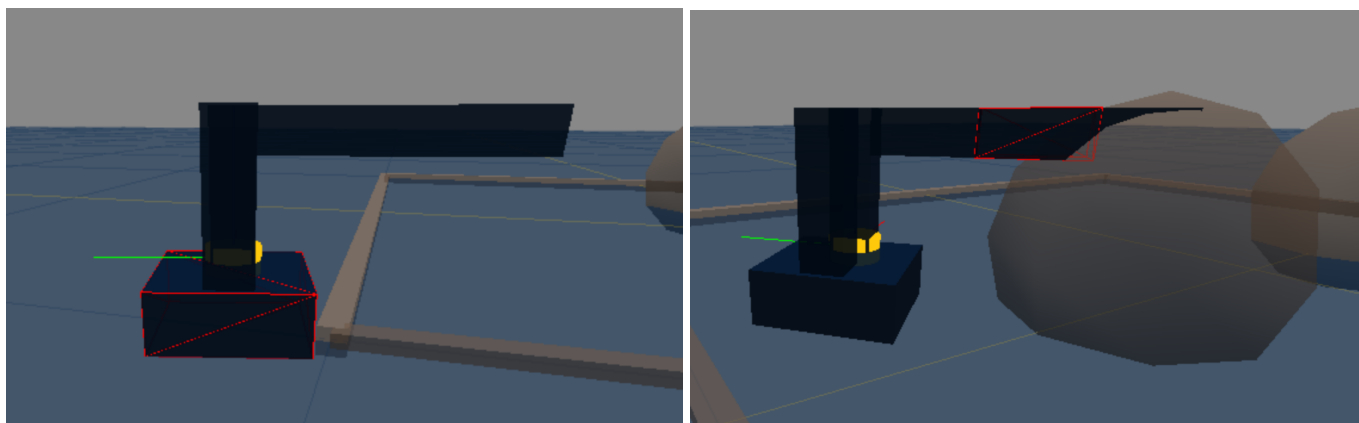
### **First step: add collision detection**

Your RRT-Connect implementation will depend on detection of collisions (provided by the function `kineval.robotIsCollision()` in `kineval_collision.js`) with respect to a specified world geometry. Worlds are specified as a rectangular boundary and sphere obstacles. A collection of worlds are provided in the "worlds/" subdirectory of `kineval_stencil`. The collision detection system performs two forms of tests: 1) testing of the base position of the robot against the rectangular extents of the world, which is provided by default, and 2) testing of link geometries for a robot configuration against spherical objects, which depends on code you will write. Collision testing for links in a configuration is performed by AABB/Sphere tests that require the bounding box of each link's geometry in the coordinates of that link. This bounding box is computed by the following within the loop inside `kineval.initRobotLinksGeoms()` in `kineval.js`:

```
// bounding box of robot link in local link coordinates
robot.links[x].bbox = new THREE.Box3;
robot.links[x].bbox =
  robot.links[x].bbox.setFromObject(robot.links[x].geom);
```

Even before your planner is implemented, you can use the collision system interactively with your robot. The provided `kineval.robotIsCollision()` function will test the current configuration of the robot. When the robot is detected to be in collision,

one of the colliding links will be highlighted with a red wireframe. There could be many links in collision, but only one will be highlighted.



The call to `kineval.robotIsCollision()` has been placed within `my_animate()` in `home.html`:

```
// show if robot is currently in collision
kineval.robotIsCollision();
```

## Updating `kineval_collision` for your implementation

To complete the collision system, you will need to modify the forward kinematics calls in `kineval/kineval_collision.js`. Specifically, you will need to perform a traversal of the forward kinematics of the robot for an arbitrary robot configuration within the function `kineval.poselsCollision()`. `kineval.poselsCollision()` takes in a vector in the robot's configuration space and returns either a boolean `false` for no detected collision or a string with the name of a link that is in collision. As a default, this function performs base collision detection against the extents of the world. For collision detection of each link, this function will make a call to function that you create called `robot_collision_forward_kinematics()` to recursively test for collisions along each link. Your collision FK recursion should use the link collision function, `collision_FK_link()`, provided below along with a joint traversal function properly positions the link and joint frames for the given configuration.

```
function collision_FK_link(link,mstack,q) {

  // this function is part of an FK recursion to test each link
  // for collisions, along with a joint traversal function for
```

```

// the input robot configuration q
//
// this function returns the name of a robot link in collision
// or false if all its kinematic descendants are not in collision

// test collision by transforming obstacles in world to link space
mstack_inv = numeric.inv(mstack);
// (alternatively) mstack_inv = matrix_invert_affine(mstack);

var i; var j;

// test each obstacle against link bbox geometry
// by transforming obstacle into link frame and
// testing against axis aligned bounding box
for (j in robot_obstacles) {

  var obstacle_local =
    matrix_multiply(mstack_inv,robot_obstacles[j].location);

  // assume link is in collision as default
  var in_collision = true;

  // return false if no collision is detected such that
  // obstacle lies outside the link extents
  // along any dimension of its bounding box
  if (
    (obstacle_local[0][0]<
      (link.bbox.min.x-robot_obstacles[j].radius)
    )
    ||
    (obstacle_local[0][0]>
      (link.bbox.max.x+robot_obstacles[j].radius)
    )
  )
    in_collision = false;

  if (
    (obstacle_local[1][0]<
      (link.bbox.min.y-robot_obstacles[j].radius)
    )
    ||
    (obstacle_local[1][0]>
      (link.bbox.max.y+robot_obstacles[j].radius)
    )
  )
    in_collision = false;

  if (
    (obstacle_local[2][0]<
      (link.bbox.min.z-robot_obstacles[j].radius)
    )

```

```

    ||
    (obstacle_local[2][0]>
      (link.bbox.max.z+robot_obstacles[j].radius)
    )
  )
  in_collision = false;

  // return name of link for detected collision if
  //   obstacle lies within the link extents
  //   along all dimensions of its bounding box
  if (in_collision)
    return link.name;
}

// recurse child joints for collisions,
//   returning name of descendant link in collision
//   or false if all descendants are not in collision
if (typeof link.children !== 'undefined') {
  var local_collision;
  for (i in link.children) {
    // STUDENT: create this joint FK traversal function
    local_collision =
      collision_FK_joint(robot.joints[link.children[i]],mstack,q)
    if (local_collision)
      return local_collision;
  }
}

// return false, when no collision detected for this link and children
return false;
}

```

kineval\_collision.js uses matrix and quaternion calls based on the reference implementation (i.e., the instructor's code). Your matrix and quaternion calls likely have a different structure to the function arguments and returned data structures. You should either:

- modify calls to matrix/quaternion routines to fit your functions, or
- use a modified version of your own FK with the collision test added to the link traversal function (remember: you need the inverse of the matrix stack for collision testing in a link frame)

You can feel free to implement `matrix_invert_affine()` instead of using `numeric.inv()`. Affine transforms can be inverted (in constant time, Quiz 3!) through a much simpler

process than the generic matrix inversion, which is  $O(n^3)$  for Gaussian elimination.

If successful to this point, you should be able to see the collision world of the robot, move around this world, and see the colliding link display a red wireframe when a collision occurs.

## Implementing and invoking the planner

Your motion planner will be implemented in the file `kineval/kineval_rrt_connect.js` through the functions `kineval.robotRRTPlannerInit()` and `robot_rrt_planner_iterate()`. The `kineval.robotRRTPlannerInit()` function should be modified to initialize the RRT trees and other necessary variables. The `robot_rrt_planner_iterate()` function should be modified to perform a **single** RRT-Connect iteration based on the current RRT trees. Basic RRT tree support functions are provided for initialization, adding configuration vertices (which renders "breadcrumb" indicators of base positions explored), and adding graph edges between configuration vertices. This function should **not** use a for loop to perform multiple planning iterations, as this will cause the browser to block and become unresponsive. Instead, the planner will be continually called asynchronously by the code stencil until a motion plan solution is found.

Once implemented, your planner will be invoked interactively by first moving the robot to an arbitrary non-colliding configuration in the world and then pressing the "m" key. The "m" key will request the generation of a motion plan. While the planner is working, it will not accept new planning requests. Thus, you can move the robot around while the planner is executing.

## Planner output

The output of your planner will be a motion path in a sequentially ordered array (named `kineval.motion_plan[]`) of RRT vertices. Each element of this array contains a reference to an RRT vertex with a robot configuration (`.vertex`), an array of edges (`.edges`), and a threejs indicator geometry (`.geom`). Once a viable motion plan is found, this path can be highlighted by changing the color of the RRT vertex "breadcrumb" geom indicators. The color of any configuration breadcrumb indicator in a tree can be modified, such as in the following example for red:

```
tree.vertices[i].geom.material.color = {r:1,g:0,b:0};
```

The user should be able to interactively move the robot through the found plan. Stencil code in `user_input()` within `kineval_userinput.js` will enable the "n" and "b" keys to move the robot to the next and previous configuration in the found path, respectively. These user key presses will respectively increment and decrement the parameter `kineval.motion_plan_traversal_index` such that the robot's current configuration will become:

```
kineval.motion_plan[kineval.motion_plan_traversal_index]
```

Note: we are **NOT** using `robot.controls` to execute the found path of the robot. Although this can be done, the collision system does not currently test for configurations that occur due to the motion between configurations.

## Testing



Make sure to test all provided robot descriptions from a reasonable set of initial configurations within all of the provided worlds, ensuring that:

- a valid non-colliding path is found and can be traversed,
- the robot does not take steps longer than 1 unit,
- the robot base does not move outside the X-Z plane. Specifically, the base should not translate along the Y axis, and should not rotate about the X and Z axes.

## Graduate Section Requirement

In addition to the requirements above, students in the graduate section must also implement the [RRT-Star](#) motion planning algorithm and test to ensure:

- joint limits for the different joint types are respected, and
- the "fetch" robot should be able to navigate all of the provided worlds.

### ADVANCED EXTENSION

Of the 4 possible advanced extension points, one additional point for this assignment can be earned by adding the capability of motion planning to an arbitrary robot configuration goal.

Of the 4 possible advanced extension points, two additional points for this assignment can be earned by using the A-star algorithm for base path planning in combination with RRT-Connect for arm motion planning.

Of the 4 possible advanced extension points, one additional point for this assignment can be earned by writing a collision detection system for two arbitrary triangles in 2D using a JavaScript/HTML5 canvas element.

Of the 4 possible advanced extension points, two additional points for this assignment can be earned by writing a collision detection system for two arbitrary triangles in 3D using JavaScript/HTML5 and threejs or a canvas element.

Of the 4 possible advanced extension points, four additional points for this assignment can be earned by implementation of triangle-triangle tests for collision detection between robot and planning scene meshes.

Of the 4 possible advanced extension points, three additional points for this assignment can be earned by implementation of cubic or quintic polynomial interpolation (Spong Ch. 5.5.1 and 5.5.2) across configurations returned in a computed motion plan.

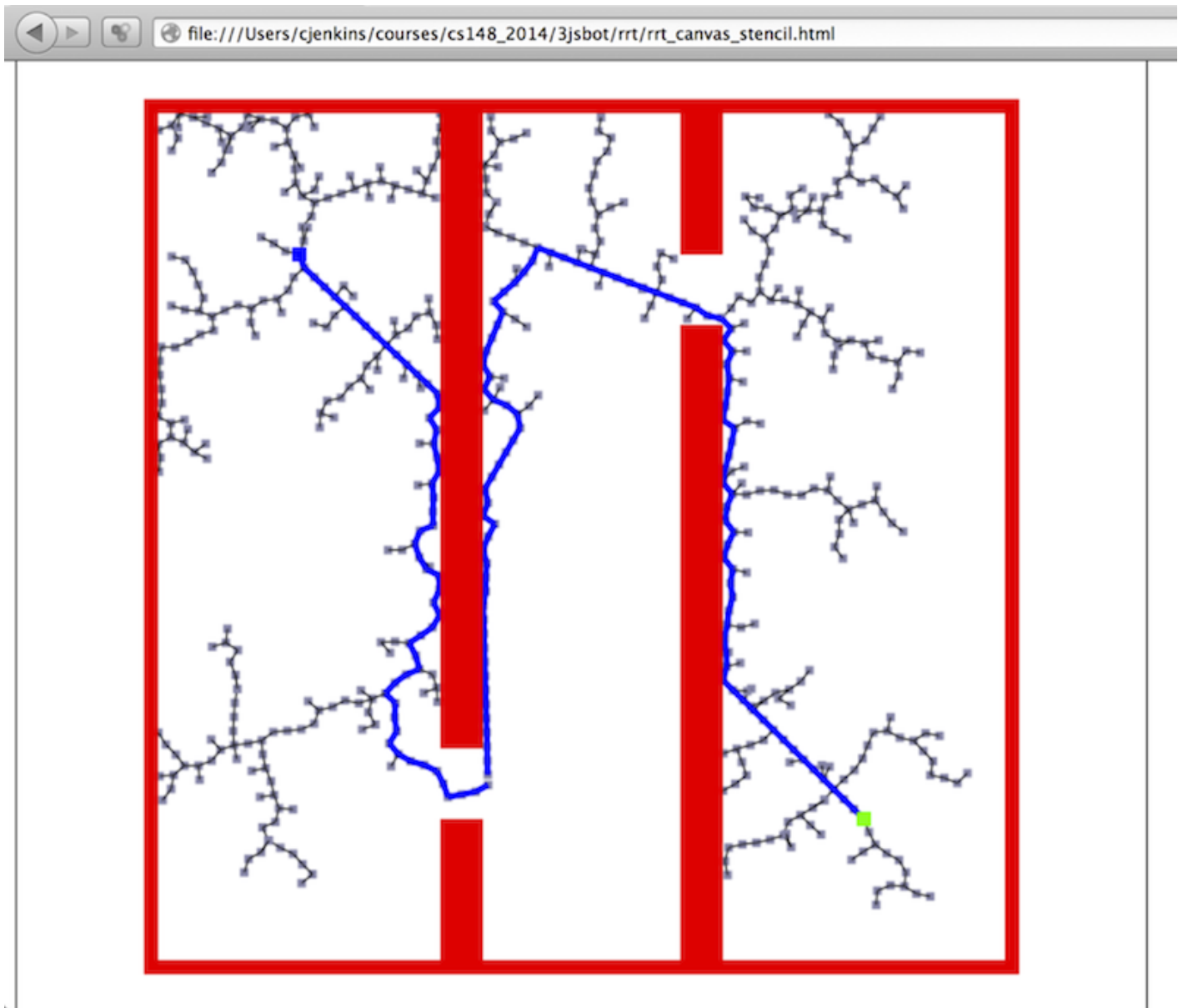
Of the 4 possible advanced extension points, four additional points for this assignment can be earned by implementing an approved research paper describing a motion planning algorithm.

**Warning: Respect configuration space**

The planner should produce a collision-free path in configuration space (over all robot DOFs) and not just the movement of the base on the ground plane. If your planner does not work in configuration space, it is sure to fail tests used for grading.

**Highly recommended: start with HTML5 Canvas Stencil**





Using the browser for as a development environment has many benefits. However, when coding mistakes occur, it will make the browser lock up and be completely unusable. Such mistakes can be especially difficult to debug when the overhead of rendering with threejs is involved.

To help you get started, the path planning code stencil in the "search\_canvas" directory has entry points for developing your core RRT routines. This stencil will allow you to implement the RRT-Connect algorithm in simplified 2D worlds with provided routines for visualization and collision. Because the RRT is invariant across configuration spaces, an RRT developed for the 2D Canvas world should easily port to the N-D threejs world, with minor changes for invoking drawing routines.

## **Project Submission**

For turning in your assignment, ensure your completed project code has been committed and pushed to the *master* branch of your repository.

## Assignment 7: The best use of robotics?

**Slides due 11:59pm, Friday, December 6, 2019**

**Presentation due 1:30pm, Monday, December 9, 2019**

Scenario: An investor is considering giving you 20 million dollars (cold hard USD cash, figuratively). This investor has been impressed by your work with KinEval and other accomplishments while at the University of Michigan. They are convinced you have the technical ability to make a compelling robot technology... but, they are unsure how this technology could produce something useful. Your task is to make a convincing pitch for a robotics project that would yield a high return on investment, as measured by some metric (financial profit, good for society, creation of new knowledge, etc.).

You will get 2 minutes to make a pitch to develop something useful with robots. Consider the instructor and your classmates as the people that need to be convinced. As a guideline, your pitch should address an opportunity (presented by a need or a problem), your planned result (as a system, technology, product, and/or service), and how you will measure successful return on investment. Return on investment can be viewed as financial profit (wrt. venture capital), good for society (wrt. a government program), creation of new knowledge or capabilities (wrt. a grant for scientific research). Remember, the purpose is to convince and inspire about what is possible, rather than dive into specifics.

The last scheduled class period and a little more (December 9, 1:30-4:30pm) will be dedicated to student presentations to pitch ideas on the best use of robotics.

Please post your slides to the "#asgn7-best-use" discussion channel before 11:59pm on Friday December 6th. Your first slide must include the title of your presentation, your name, and your username. The filename of your slides must start with your username in the format "asgn7\_username". Slides will only be accepted in PDF format, although embedding of videos or links to videos will be accepted. You can post

new versions of your slides up to the submission deadline on December 6th, but must delete older versions.

The pitch judged to be the most convincing will get first dibs.

## Additional Materials

### Appendix: Git-ing Started with Git

Using version control effectively is an essential skill for both the AutoRob course and, more generally, contributing to advanced projects in robotics research and development. git is arguably the most widely used version control system at current. Examples of the many robotics projects using git include: [Lightweight Communications and Marshalling](#), [the Robot Operating System](#), [Robot Web Tools](#), [Fetch Robotics](#), [the NASA Robonaut 2](#), and [the Rethink Baxter](#). To help you use git effectively, the course staff has added the tutorials below for getting started with git. This is meant to be a starting guide to using git version control and the bash command shell. For a more complete list of commands and features of git, you can refer to the following guides: [The Git Pro book](#) or The [Basic git command line reference for windows users](#). An interactive tutorial for git is available at [LearnGitBranching](#).

#### Installing git

The AutoRob course assumes git is used from an command line terminal to work with a git hosting service, such as [GitHub](#) or [Bitbucket](#). Such terminal environments are readily available in Linux and Mac OSX through their respective terminal programs. For MS Windows, we are recommending Git Bash, which can be downloaded from the [Git for Windows](#) project. Several other viable alternatives git clients exist, such as the GUI-based [GitKraken](#).

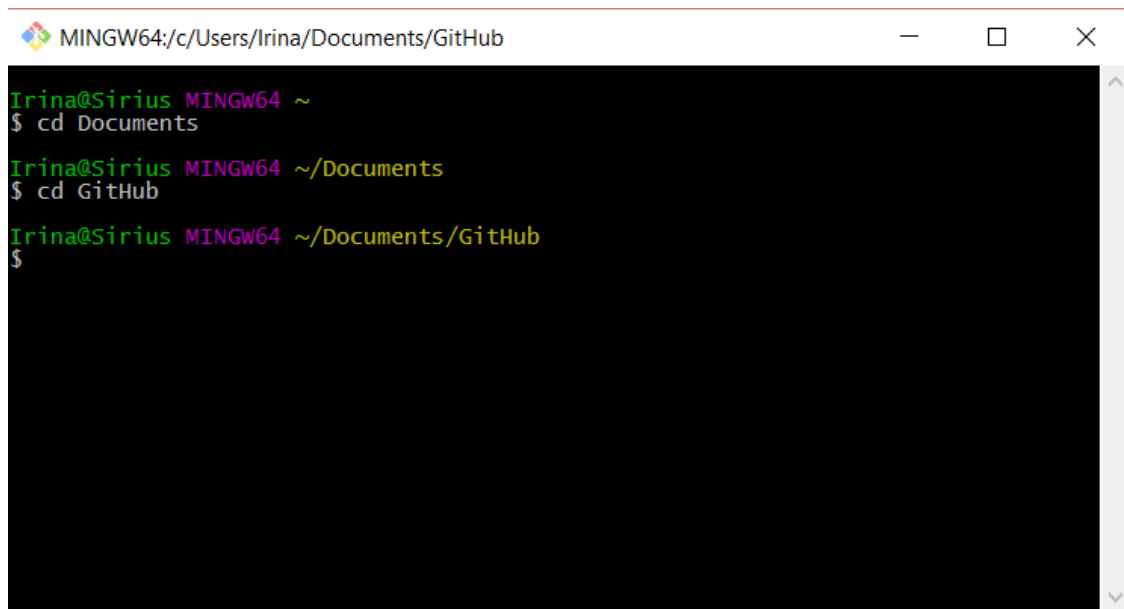
git can be installed on Linux through a common package management system, based on your distribution, with one of the following commands:

```
sudo yum install git-all
```

```
sudo apt-get install git-all
```

For Mac OSX, git can be installed on its own using the [Git-OSX-Installer](#) or as part of larger set of [Xcode](#) build tools.

If you have a command line terminal running, you should see a shell environment that looks something like this (screenshot from an older version of Git Bash):



```
MINGW64:/c/Users/Irina/Documents/GitHub
Irina@Sirius MINGW64 ~
$ cd Documents
Irina@Sirius MINGW64 ~/Documents
$ cd GitHub
Irina@Sirius MINGW64 ~/Documents/GitHub
$
```

If you have git installed, you should be able to enter the "git" command and see the following usage information printed (screenshot from OSX):

```
10. bash
Last login: Mon Jan 18 13:50:50 on ttys009
0587358173:~ logan$ git
usage: git [--version] [--help] [-C <path>] [-c name=value]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
        [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  reset      Reset current HEAD to the specified state
  rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
  bisect     Use binary search to find the commit that introduced a bug
  grep       Print lines matching a pattern
  log        Show commit logs
  show       Show various types of objects
  status     Show the working tree status

grow, mark and tweak your common history
  branch     List, create, or delete branches
  checkout   Switch branches or restore working tree files
  commit     Record changes to the repository
  diff       Show changes between commits, commit and working tree, etc
  merge      Join two or more development histories together
  rebase     Forward-port local commits to the updated upstream head
  tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
  fetch      Download objects and refs from another repository
  pull       Fetch from and integrate with another repository or a local branch
  push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
0587358173:~ logan$
```

## Cloning your repository

The most common thing that you will need to do is pull and push files from and to your git hosting service. Upon opening Git Bash, you will need to go to the location of **both** your GitHub/Bitbucket repository on the web and your git workspace on your local computer. Our first main step is to clone your remote repository onto your local computer. Towards this end, the next step is to open your terminal application and determine your current directory, assuming you will use this directory to create a workspace. For Linux and OSX, the terminal should start in your home directory, often

"/home/username" or "/Users/username". For Git Bash on Windows, the default home directory location could be the Documents in your user directory, or the general user folder within "C:\Users".

From your current directory, you can use Bash commands to view and modify the contents of directories and files. You can see a list of the files and folders that can be accessed using ls (list) and change the folder using the command cd (change directory) as shown below. If you believe that the directory has files in addition to folders, but would like a list of just the folders, then the command ls -d \*/ can be used instead of ls. Below is a quick summary of relevant Bash commands:

- "ls" prints a listing of files in the current directory
- "pwd" prints the location of the current directory in the filesystem
- "cd [NameFolder]" moves the terminal to a new directory in the filesystem
- "ls [Expression]" prints a listing of files in the current directory matching the given Expression; ls r\* prints all files starting with the character 'r'
- "mkdir [NameFolder]" creates a folder within the current directory. If the folder name has spaces, then NameFolder will need to be in double quotes.
- "rmdir [NameFolder]" removes a specified empty folder. If it is not empty, the folder will not be removed.
- "rm -rf [NameFolder]" removes a specified folder and all the contents
- "touch [FileName]" creates a single empty text file
- "touch [FileName1.txt] [FileName2.txt]..." creates multiple empty text files
- "rm [FileName]" removes a specific file from the current directory
- "rm -i "rm -v [FileName]" removes the file and reports in console

You are now ready to clone a copy of your remote repository and populate it with files for AutoRob projects. It is assumed that you have already created a repository on your git hosting service, given the course staff access to this repository, and provided a link of your repository to the course staff. This repository link (in the form of "https://github.com/user\_name/repository\_name.git") will now be used to clone a copy of your remote repository onto your local machine using the following git command below. This command will clone the repository contents to a subdirectory labeled with the name of the repository:

```
git clone [repository URL link]
```

This directory should be listed and inspected to ensure it has been cloned with the contents of the repository, matching the remote repository from your git hosting service. If this is a new repository, it is not a problem for this directory to be empty:

```
ls [repository_name]
```

You can also check for differences between the files on your computer and the remote repository using `git status` as shown below. If you receive the message shown in the example below, then there are no differences. If there are differences, then it will have the number of files which are different highlighted in red.

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

### **Important: workspace is not the same as repository**

You should now have a local copy of your repository with a workspace in a subdirectory. It is critical to note that your local repository is different than the subdirectory with your current workspace. Your workspace is not automatically tracked by the version control system and considered ephemeral. Any changes made to your workspace must be committed back into the local repository to be recognized by the version control system. Further, any changes committed to your local repository must also be pushed remotely to be recognized by your git hosting service. Thus, any changes made to your workspace can be lost if not committed and pushed, which will be discussed more in later sections.

### **Populating your repository with project stencil code**

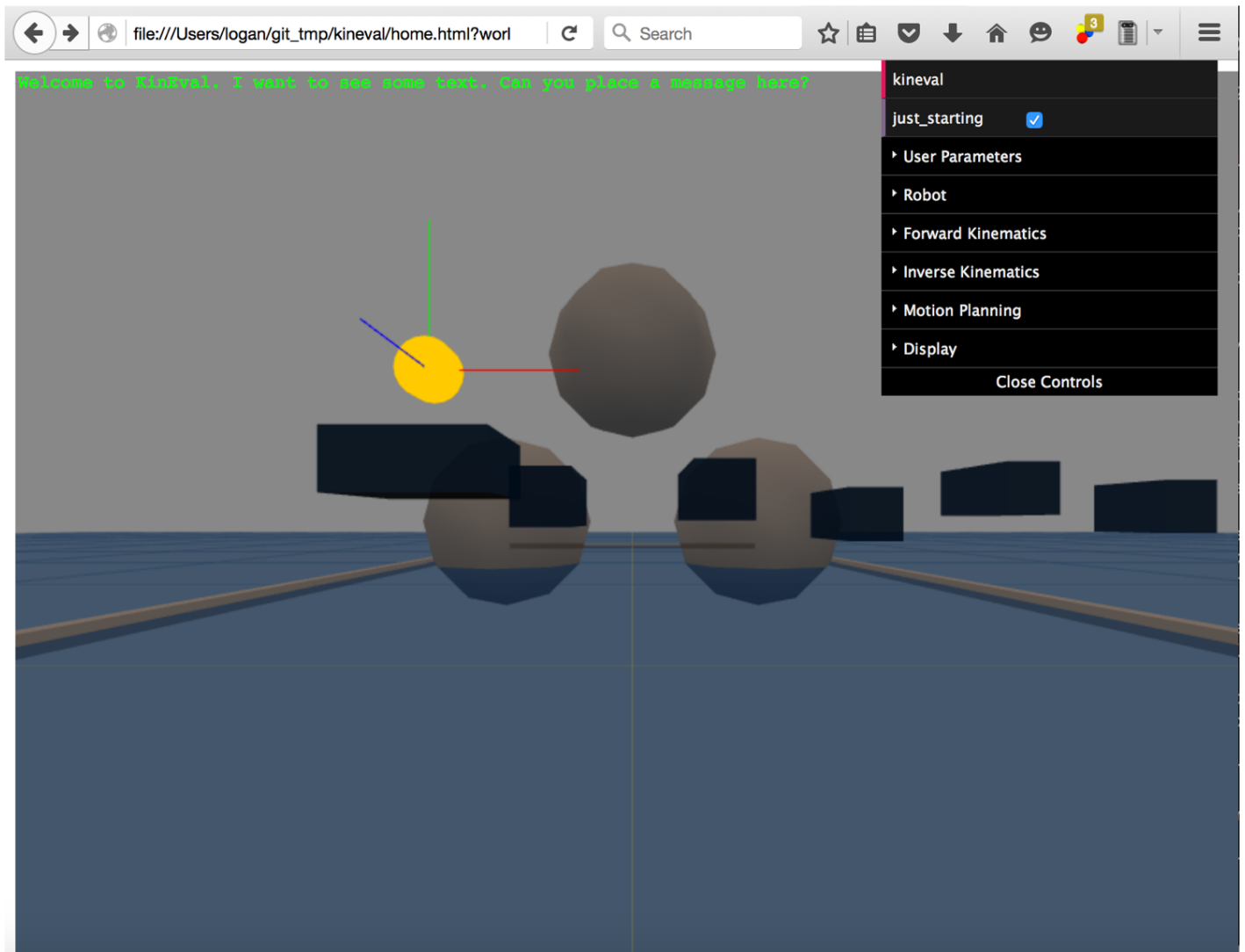
In a separate directory, clone the `kineval-stencil` repository to a subdirectory on your local machine:

```
cd [home_directory]
git clone git@github.com:autorob/kineval-stencil.git
```

Inspect this directory to ensure it has been cloned with the contents of the repository:

```
cd kineval-stencil
ls
```

and open "home.html" from this directory in a web browser and ensure you see the starting point picture below:



If your browser throws an error when loading "home.html", one potential cause is that this browser disallows loading of local files. In such cases, the browser will typically report a security error in the console. This security issue is avoided by serving the KinEval files from an HTTP server. Such a HTTP server is commonly available within distributions for modern operating systems. Assuming Python is installed on your computer, you can start a HTTP with the following command from your workspace directory, and then view the page at [localhost:8000](http://localhost:8000):

```
python -m SimpleHTTPServer
```

Alternatively, if you have nodejs installed, you can install and start a HTTP with the following command from your workspace directory, and then view the page at



localhost:3000:

```
npm install simple-server  
node simple-server
```

Once you are able to load and view "home.html", the next step is to copy the kineval-stencil files to the directory with your workspace

```
cd [home directory]  
cp -r kineval-stencil/* [repository_name]/
```

**IMPORANT:** ensure that this copy from kineval-stencil to your repository does not include the ".git" subdirectory. The ".git" subdirectory is the actual git repository on your local computer, where as the directory itself is your (ephemeral) workspace outside of git.

As these copied files are new to your working repository, they need to be added to the repository to ensure they are tracked for version control. These files are added with the following commands:

```
cd [repository name]  
git add *
```

Below is a more detailed summary of git commands for adding files from your workspace to your repository:

- "git add" adds changed files to the next commit. There are several different options which can follow this command.
- "git add -A: adds all new files and changes to the next commit including deletions
- "git add ." adds all new files and changes to the next commit without deletions
- "git add -u" adds all changes to the next commit without new files

## **Commit and push to update your repository**

Whenever you make any significant changes to your repository, these changes should be committed to your local repository and pushed to your remote repository. Such changes can involve adding new files or modifying existing files in your local workspace. For such changes, you will first commit changes from your workspace to your local repository using the git commit command:

```
git commit -a -m "message describing changes"
```

and then pushing these changes from your local repository to a synced repository on your git hosting service:

```
git push
```

This commit will occur to the "master" branch of your repository.

Note: the change files must be located in the correct repository folder on your local computer and these commands should be performed in the local workspace directory. Below is a more detailed summary of git commands for adding files from your workspace to your repository:

- "git commit" commits files with changes. There are several options that can follow this. The message text is required and is good practice to list changes that you have made. GitHub is good at tracking changes.
- "git commit [FileName] -m 'Message'" commits changes to a specific file
- "git commit -a "Message'" commits all files changed since last commit
- "git commit -a -m "Message'" commits all files changed since last commit but not new files
- "git push" pushes the committed changes to remote repository

Once you have committed and pushed, your local workspace becomes redundant as your changes have been stored and tracked remotely. The local workspace can now be deleted without concern. This local workspace can also be updated with changes to the remote repository by pulling.

## **Pulling remote changes**

Changes be made to your remote repository, potentially by other collaborators, without being tracked by your local repository. This can lead to potential versioning conflicts when committed changes contradict each other. For the AutoRob course, versioning conflicts should not be a problem because commits to your repository should be yours alone. That said, one good practice is to ensure your workspace, local repository, and remote repository are synced before making any changes. A brute force method for doing this is to re-clone your repository each time you begin to make changes. Another option is to pull remote changes into your local repository and workspace using the git pull command (or git fetch command):

```
cd [repository_name]
git pull
```

Below is a more detailed summary of git commands for pulling and fetching:

- "git pull [RemoteName]" is used for retrieving commits and merging the files to what is already on the computer. This may make changes to the files that are already there; effectively, this is a fetch followed by a merge. If you need to pull directly from the repository which already exists and has been accessed through the change directory command, then you do not list a [RemoteName].
- "git fetch " used for retrieving commits from a repository that does not already exist on the user's computer. This creates an exact copy of the files to your local repository and not to the workspace.

## Branching

Branching is an effective mechanism for work in a repository to be done in parallel with changes merged at a later point. A branch essentially creates a copy of your work at a particular version. Branches are independently tracked by the version controller and can be merged together when requested (which collaboratively results in a "pull request"). The larger story for branching and merging is outside the scope of AutoRob.

The working version of your code, which you submit for grading, is expected to be in the "*master*" branch of your repository. When working on a new assignment, it is recommend that you create a branch for this new work. This allows your stable code in the master branch to be undisturbed while you continue to modify your code. Once your work for this assignment is done, you can then update your *master* by merging in your assignment branch. Stylistically, it is helpful to use the name *Assignment-X* for your assignment branch for Project number *X*.

The simplest means for branching in this context is to use the branching feature from the webpage of your remote repository. From GitHub, simply select the master branch from the "Branch: " button and enter the name of the branch to be created. From Bitbucket, select the "Branches" icon from the left hand toolbar and follow the instructions for branch creation. If successful, you should see a list of branches that can each be inspected for their respective contents. Branches can also be deleted from this interface.

A branch can also be created from the command line by the following, which will create a copy of the current branch:

```
git branch [branch_name]
```

You can switch between branches with the following command:

```
git checkout [branch_name]
```

as well as clone a specific branch from a repository:

```
git clone -b [branch_name] [repository URL link]
```

Good luck and happy hacking!







**Course Approval Request Form**  
Office of the Registrar, University of Michigan

1210 ISA Building  
500 S. State Street  
Ann Arbor, MI 48109-1382  
Phone: 734.763.2113  
Fax: 734.936.3148  
ro.curriculum@umich.edu  
ro.umich.edu

CHECK APPROPRIATE BOXES FOR ALL CHANGES

Action Requested

- New Course  
 Modification of Existing Course  
 Deletion of Existing Course

Date of Submission: 2019-12-17  
Effective Term: Fall 2020

<input checked="" type="checkbox"/>	<b>Course Offered</b> <input checked="" type="checkbox"/> Indefinitely <input type="checkbox"/> One term only	<b>RO USE ONLY</b> Date Received: Date Completed: Completed By:
-------------------------------------	---	--

**CURRENT LISTING**

**REQUESTED LISTING**

<input type="checkbox"/>	Dept (Home): Mechanical Engineering Subject: MECHENG Catalog: 450			Dept (Home): Mechanical Engineering Subject: MECHENG Catalog: 450		
<input type="checkbox"/>	<input type="checkbox"/> Course is Cross-Listed with Other Departments			<input type="checkbox"/> Course is Cross-Listed with Other Departments		
<input type="checkbox"/>	Department	Subject	Catalog Number	Department	Subject	Catalog Number
<input type="checkbox"/>	Course Title (full title) Design and Manufacturing III			Course Title (full title) Design and Manufacturing III		
<input type="checkbox"/>	Abbreviated Title (20 char) Des/Mfg III			Abbreviated Title (20 char) Des/Mfg III		
<input checked="" type="checkbox"/>	Course Description (Please limit to 50 words and attach separate sheet if necessary) A mechanical engineering design project by which the student is exposed to the design process from concept through analysis to layout and report. Projects are proposed from the different areas of study within mechanical engineering and reflect the expertise of instructional faculty and industrial representatives.					
<input checked="" type="checkbox"/>	Full Term Credit Hours Undergraduate Min: 4      Graduate Min: Undergraduate Max: 4      Graduate Max:			Half Term Credit Hours Undergraduate Min:      Graduate Min: Undergraduate Max:      Graduate Max:		
<input checked="" type="checkbox"/>	Course Credit Type Undergraduate Student					
<input type="checkbox"/>	Repeatability <input type="checkbox"/> Course is Repeatable for Credit Maximum number of repeatable credits:			<input type="checkbox"/> Course is Y graded <input type="checkbox"/> Can be taken more than once in the same term		

Subject: Mechanical Engineering      Catalog: 450		
<input checked="" type="checkbox"/>	<b>Grading Basis</b> <input checked="" type="checkbox"/> Graded (A – E) <input type="checkbox"/> Credit/No Credit <input type="checkbox"/> Satisfactory/Unsatisfactory <input type="checkbox"/> Pass/Fail <input type="checkbox"/> Business Administration <b>Grading</b> <input type="checkbox"/> Not for Credit <input type="checkbox"/> Not for Degree Credit <input type="checkbox"/> Degree Credit Only	<b>Add Consent</b> <input type="checkbox"/> Department Consent <input type="checkbox"/> Instructor Consent <input checked="" type="checkbox"/> No Consent  <b>Drop Consent</b> <input type="checkbox"/> Department Consent <input type="checkbox"/> Instructor Consent <input checked="" type="checkbox"/> No Consent

	CURRENT LISTING	REQUESTED LISTING
<input type="checkbox"/>	Advisory Prerequisite (254 char)	Advisory Prerequisite (254 char)
<input checked="" type="checkbox"/>	Enforced Prerequisite (254 char) MECHENG 350 and 360 and (MECHENG 395 or AEROSP 305); (C- or better) Minimum grade requirement:	Enforced Prerequisite (254 char) MECHENG 320 and 350 and 360 and (MECHENG 395 or AEROSP 305); (C- or better) Minimum grade requirement:
<input type="checkbox"/>	Credit Exclusions May not be taken concurrently with ME455 or ME495. Not open to graduate students.	Credit Exclusions May not be taken concurrently with ME455 or ME495. Not open to graduate students.
<input checked="" type="checkbox"/>	<b>Course Components</b> <input checked="" type="checkbox"/> Lecture <input type="checkbox"/> Seminar <input type="checkbox"/> Recitation <input checked="" type="checkbox"/> Lab <input type="checkbox"/> Discussion <input type="checkbox"/> Independent Study	<b>Graded Component</b> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
		<b>Terms Typically Offered</b> <input checked="" type="checkbox"/> Fall <input checked="" type="checkbox"/> Winter <input type="checkbox"/> Spring <input type="checkbox"/> Summer <input type="checkbox"/> Spring/Summer
Cognizant Faculty Member Name: Steve Skerlos      Cognizant Faculty Member Title:		

**SIGNATURES ARE REQUIRED FROM ALL DEPARTMENTS INVOLVED (Please Print AND Sign Name)**

Contact Person:	Email:	Phone:
Curriculum Committee Member:	Print:	Date:
Curriculum Committee Chair:	Print:	Date:
Home Department Chair: <i>[Signature]</i>	Print: <i>Kenn Oldham</i>	Date: <i>12/17/19</i>
Cross-Listed Department Chair:	Print:	Date:
Cross-Listed Department Chair:	Print:	Date:
Cross-Listed Department Chair:	Print:	Date:

**DEPARTMENTAL/COLLEGE USE ONLY**

**Current:**

Course Description

A mechanical engineering design project by which the student is exposed to the design process from concept through analysis to layout and report. Projects are proposed from the different areas of study within mechanical engineering and reflect the expertise of instructional faculty and industrial representatives.

Class Length

Full term

Contact hours (lecture):

Contact hours (recitation)

Contact hours (lab)

**Requested:**

Course Description

A mechanical engineering design project by which the student is exposed to the design process from concept through analysis to layout and report. Projects are proposed from the different areas of study within mechanical engineering and reflect the expertise of instructional faculty and industrial representatives.

Class Length

Full term

Contact hours (lecture):

Contact hours (recitation)

Contact hours (lab)

**Additional Info:**

Submitted by:

Home dept

Describe how this course fits with the degree requirements:

Core Course

ABET departmental program outcomes for undergraduate courses:

1,2,3,4,5,6,7

Special resources of facilities required for this course:

Supporting statement:

Change of adding MECHENG 320 as a prerequisite is made to ensure that students enter ME450 with same courses completed as they would have prior to change in ME395 prerequisites





**Course Approval Request Form**  
Office of the Registrar, University of Michigan

1710 ISA Building  
500 S. State Street  
Ann Arbor, MI 48109-1382  
Phone: 734.763.2113  
Fax: 734.936.3148  
ro.curriculum@umich.edu  
ro.umich.edu

CHECK APPROPRIATE BOXES FOR ALL CHANGES

Action Requested

- New Course  
 Modification of Existing Course  
 Deletion of Existing Course

Date of Submission: 2019-12-17  
Effective Term: Fall 2020

<input checked="" type="checkbox"/>	Course Offered <input checked="" type="checkbox"/> Indefinitely <input type="checkbox"/> One term only	<b>RO USE ONLY</b> Date Received: Date Completed: Completed By:
-------------------------------------	--	--

**CURRENT LISTING**

**REQUESTED LISTING**

<input type="checkbox"/>	Dept (Home): Mechanical Engineering Subject: MECHENG Catalog: 395	Dept (Home): Mechanical Engineering Subject: MECHENG Catalog: 395												
<input type="checkbox"/>	<input type="checkbox"/> Course is Cross-Listed with Other Departments	<input type="checkbox"/> Course is Cross-Listed with Other Departments												
<input type="checkbox"/>	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 20%;">Department</th> <th style="width: 20%;">Subject</th> <th style="width: 60%;">Catalog Number</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> <td> </td> </tr> </tbody> </table>	Department	Subject	Catalog Number				<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 20%;">Department</th> <th style="width: 20%;">Subject</th> <th style="width: 60%;">Catalog Number</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> <td> </td> </tr> </tbody> </table>	Department	Subject	Catalog Number			
Department	Subject	Catalog Number												
Department	Subject	Catalog Number												
<input type="checkbox"/>	Course Title (full title) Laboratory I	Course Title (full title) Laboratory I												
<input type="checkbox"/>	Abbreviated Title (20 char) Lab I	Abbreviated Title (20 char) Lab I												
<input checked="" type="checkbox"/>	Course Description (Please limit to 50 words and attach separate sheet if necessary) Weekly lectures and experiments designed to introduce the student to the basics of experimentation, instrumentation, data collection and analysis, error analysis, and reporting. Topics will include fluid mechanics, thermodynamics, mechanics, materials, and dynamical systems. Emphasis is placed on report writing and team-building skills.													
<input type="checkbox"/>	<b>Full Term Credit Hours</b> Undergraduate Min:      Graduate Min: Undergraduate Max:      Graduate Max:	<b>Half Term Credit Hours</b> Undergraduate Min:      Graduate Min: Undergraduate Max:      Graduate Max:												
<input checked="" type="checkbox"/>	Course Credit Type Undergraduate Student													
<input type="checkbox"/>	Repeatability <input type="checkbox"/> Course is Repeatable for Credit Maximum number of repeatable credits:	<input type="checkbox"/> Course is Y graded <input type="checkbox"/> Can be taken more than once in the same term												

Subject: Mechanical Engineering Catalog: 395

<input checked="" type="checkbox"/>	<b>Grading Basis</b> <input checked="" type="checkbox"/> Graded (A – E) <input type="checkbox"/> Credit/No Credit <input type="checkbox"/> Satisfactory/Unsatisfactory <input type="checkbox"/> Pass/Fail <input type="checkbox"/> Business Administration <b>Grading</b> <input type="checkbox"/> Not for Credit <input type="checkbox"/> Not for Degree Credit <input type="checkbox"/> Degree Credit Only	<b>Add Consent</b> <input type="checkbox"/> Department Consent <input type="checkbox"/> Instructor Consent <input checked="" type="checkbox"/> No Consent	<b>Drop Consent</b> <input type="checkbox"/> Department Consent <input type="checkbox"/> Instructor Consent <input checked="" type="checkbox"/> No Consent
-------------------------------------	---	--	---

**CURRENT LISTING**


**REQUESTED LISTING**

<input type="checkbox"/>	Advisory Prerequisite (254 char)	Advisory Prerequisite (254 char)
<input checked="" type="checkbox"/>	Enforced Prerequisite (254 char) [PHYSICS 240 or 260; (C or better)] AND [PHYSICS 241 or 261; C or better] AND [MECHENG 211 and 235 and 240; (C or better) AND [Co-req of MECHENG 320 and 382] Minimum grade requirement:	Enforced Prerequisite (254 char) [PHYSICS 240 or 260; (C or better)] AND [PHYSICS 241 or 261; C or better] AND [MECHENG 211 and 235 and 240; (C or better) AND [Co-req of 382] Minimum grade requirement:
<input type="checkbox"/>	Credit Exclusions	Credit Exclusions

<input checked="" type="checkbox"/>	<b>Course Components</b> <input checked="" type="checkbox"/> Lecture <input type="checkbox"/> Seminar <input type="checkbox"/> Recitation <input checked="" type="checkbox"/> Lab <input type="checkbox"/> Discussion <input type="checkbox"/> Independent Study	<b>Graded Component</b> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	<b>Terms Typically Offered</b> <input checked="" type="checkbox"/> Fall <input checked="" type="checkbox"/> Winter <input type="checkbox"/> Spring <input type="checkbox"/> Summer <input type="checkbox"/> Spring/Summer
-------------------------------------	--	--	--

Cognizant Faculty Member Name: Volker Sick Cognizant Faculty Member Title:

**SIGNATURES ARE REQUIRED FROM ALL DEPARTMENTS INVOLVED (Please Print AND Sign Name)**

Contact Person:	Email:	Phone:
Curriculum Committee Member:	Print:	Date:
Curriculum Committee Chair:	Print:	Date:
Home Department Chair: 	Print: Kenn Oldham	Date: 12/17/19
Cross-Listed Department Chair:	Print:	Date:
Cross-Listed Department Chair:	Print:	Date:
Cross-Listed Department Chair:	Print:	Date:

**DEPARTMENTAL/COLLEGE USE ONLY**

**Current:**

Course Description

Weekly lectures and experiments designed to introduce the student to the basics of experimentation, instrumentation, data collection and analysis, error analysis, and reporting. Topics will include fluid mechanics, thermodynamics, mechanics, materials, and dynamical systems. Emphasis is placed on report writing and team-building skills.

Class Length

Full term

Contact hours (lecture):

Contact hours (recitation)

Contact hours (lab)

**Requested:**

Course Description

Weekly lectures and experiments designed to introduce the student to the basics of experimentation, instrumentation, data collection and analysis, error analysis, and reporting. Topics will include fluid mechanics, thermodynamics, mechanics, materials, and dynamical systems. Emphasis is placed on report writing and team-building skills.

Class Length

Full term

Contact hours (lecture):

Contact hours (recitation)

Contact hours (lab)

**Additional Info:**

Submitted by:

Home dept

Describe how this course fits with the degree requirements:

Core Course

ABET departmental program outcomes for undergraduate courses:

1,3,4,5,6

Special resources of facilities required for this course:

Supporting statement:

Change is intended to reflect realistic timing of fluid dynamics content in ME 395 laboratories, and reduce pressure on ME 320 instructors to reach specific topics at specific times. Change may also reduce challenges with staying on track for ME 395 pre-/co-requisites by junior year in ME curriculum, and better balance Fall/Winter enrollments in ME320, ME395, and ME350.