
Stanford Pupper

Release 2021

Nathan Kau, Abi Lopez, Gabrael Levine

Sep 20, 2023

COURSE INFORMATION

1	Course overview
---	-----------------

1

COURSE OVERVIEW



Fig. 1: Pupper Robot

A hands-on introduction to building AI-enabled robots.

2023 Teaching team:

- Head teacher: Gabrael Levine (CS 2023, Stanford Student Robotics)
- Head teacher: Jaden Clark (CS 2024, Stanford Student Robotics)
- Lead TA: Mark Leone (ME 2024, Stanford Student Robotics)
- Stuart Bowers (Hands-On Robotics)

- Jie Tan (Google DeepMind)

Guest Lecturers:

- Tingnan Zhang (Google DeepMind)
- Adrien Gaidon (Toyota Research Institute)

Overview:

Welcome to the course page for Stanford Student Robotics' course in legged robots. The course has been taught at Stanford, UW, WashU, Foothills College, Brandeis, and George Mason, with plans to expand worldwide!

In the first six weeks, students will learn key robotics concepts like including motor control, forward and inverse kinematics, and system identification; as well as important embodied-AI concepts including reinforcement learning and simulation. Through weekly labs, students will build a pair of teleoperated robot arms with haptic feedback, program a robot arm to learn to move by itself, and most importantly, build and program an agile robot quadruped called Pupper (pictured above). In the last four weeks of the course, students will pursue an open-ended project using Pupper as a platform, such as teaching Pupper to walk using reinforcement learning, building a vision system so Pupper can play fetch, or redesigning the hardware to make the robot more agile.

Researchers from Google DeepMind and Toyota Research Institute will give guest lectures during the quarter on their work teaching robots new skills using reinforcement learning.

"Empowering robots with AI is essential to make them smart and useful in people's daily life. It is one of the most important research directions in both academia and industry. This class teaches the most relevant skills, gives students hand-on experiences, and prepares them for a career in the area of AI and robotics." - Jie Tan, Staff Research Scientist at Google DeepMind

Expected time commitment: 6 - 8 hours per week.

Estimated class size: 8 - 15 students

Prerequisites: CS106B or similar coding experience strongly recommended. Coding will be majority Python but some C++ (Arduino). Familiarity with the command line. Math 51 or CME 100 or equivalent understanding of gradients. No robotics experience necessary!!

Grading: Pass/Fail for 2 or more units. Grading based on participation.

Spring Quarter Faculty sponsor: Professor Karen Liu

1.1 Syllabus

A hands-on introduction to building AI-enabled robots.

Teaching team:

- Nathan Kau (MSME 2022, Stanford Student Robotics)
- Gabrael Levine (CS 2023, Stanford Student Robotics)
- Abdulwahab Omria (CS 2023, Stanford Student Robotics)
- Raghav Samavedam (CS 2024, Stanford Student Robotics)
- Stuart Bowers (Hands-On Robotics)
- Jie Tan (Google Brain)

Overview:

In the first six weeks, students will learn key robotics concepts like including motor control, forward and inverse kinematics, and system identification; as well as important embodied-AI concepts including reinforcement learning



Fig. 2: Pupper Robot

and simulation. Through weekly labs, students will build a pair of teleoperated robot arms with haptic feedback, program a robot arm to learn to move by itself, and most importantly, build and program an agile robot quadruped called Pupper (pictured above). In the last four weeks of the course, students will pursue an open-ended project using Pupper as a platform, such as teaching Pupper to walk using reinforcement learning, building a vision system so Pupper can play fetch, or redesigning the hardware to make the robot more agile.

Researchers from Google Brain will give a guest lecture during the quarter on their work teaching robots new skills using reinforcement learning.

“Empowering robots with AI is essential to make them smart and useful in people’s daily life. It is one of the most important research directions in both academia and industry. This class teaches the most relevant skills, gives students hand-on experiences, and prepares them for a career in the area of AI and robotics.” - Jie Tan, Staff Research Scientist at Google Brain

Expected time commitment: 6 - 8 hours per week.

Estimated class size: 8 - 12 students

Prerequisites: CS106B or similar coding experience strongly recommended. Coding will be majority Python but some C++ (Arduino). Familiarity with the command line. Math 51 or CME 100 or equivalent understanding of gradients. No robotics experience necessary!!

Grading: Pass/Fail for 2 or more units. Grading based on participation.

Spring Quarter Faculty sponsor: Professor Karen Liu

1.2 Schedule

Week 1

- Lecture on BLDC robot actuators and low-level control of robot joints.
- `../course-material/lab-1` Introduction to Teensy microcontroller. Mechatronics assembly. Experimenting with PID to move BLDC actuators to desired angles. Moving three actuators simultaneously.
- [Lab 2 - Bad Robot Surgeon](#) Build 3-DOF leader-follower robot arms akin to a surgical robotic system.

Week 2

- Lecture on forward kinematics of open-chain robots and applications.
- `../course-material/lab-3` Program the robot to tell you the cartesian coordinates of the leg for any given position. Staying within a cartesian-space safety box with haptic feedback.

Week 3

- Lecture on inverse kinematics of open-chain robots and applications
- `../course-material/lab-4` Program legs to move to desired locations using inverse kinematics. Finally, drawing images with a robot arm! Task-space impedance control if time allows.

Week 4

- Lecture on quadruped gaits: e.g. theory behind gallop, walk, trot, etc and associated stability.
- `../course-material/lab-5` Program the robot to trot using open-loop task-space trajectories. Simulating the robot in PyBullet to design and test code before deploying in the real world.

Week 5

- `../course-material/lab-6` Students build full Pupper robot.

Week 6

- Survey of topics to explore in the final project including reinforcement learning, computer vision, hardware redesign, human interaction.
- Google Brain guest lecture.
- Students brainstorm project ideas, do background research, and determine final project.

Week 7

- Mini-lectures on any additional material students might find useful for their project.
- Open lab time to work on final projects

Week 8

- Students present project prototype

Week 9

- Open lab time

Week 10

- Students finish final project and showcase to faculty and Google Brain.

Week 11 (finals week)

- Students upload code, documentation, and media so that future students / hobbyists can recreate their work.
- Play with robots!

1.3 Sourcing Course Materials

1.3.1 Bill of Materials

Google Spreadsheet

If you'd like to just do Labs 1 through 5 with the robot arms, go to the tab called Arm Labs BOM to see what you need to purchase. Otherwise if you'd like to build the full Pupper robot, use the Pupper v2.1 BOM tab.

1.3.2 Overview

The parts for Pupper v2.1 are sourced from four main vendors: Amazon, DJI, JLCPCB, and McMaster.

Ordering the parts from Amazon, DJI, and McMaster is very straightforward. However, the robot uses two custom PCBs which are slightly more complicated to order. The basic procedure is to upload our PCB files to JLCPCB, which will make the boards and assemble the components onto the boards for you. Detailed instructions are linked in the BOM.

The robot also uses many 3D printed parts, which we usually print ourselves on hobby FDM printers. Instructions are linked in the BOM.

1.3.3 Notes

Sometimes the BMI160 IMU is out of stock. In that case, leave it off the bottom PCB and you can drop in a [ICM20649](#).

1.4 Lab 1 - Hello PID

Contents

- *Lab 1 - Hello PID*
 - *Mini-lecture - Actuators and PD Control*
 - *Lab Instructions*
 - *(Old) Mini-lecture - Joint Control*

1.4.1 Mini-lecture - Actuators and PD Control

Actuators and PD control lecture by Stuart Bowers, April 10, 2023.

1.4.2 Lab Instructions

Goal: Build two robot arms that mirror each other's motion.

[Insert GIF of completed robot arms]

Step 0. Setup

1. Install [VSCode](#)
2. Install the PlatformIO extension for VSCode. (Might take several minutes on Windows - check the bottom bar in VSCode for status)

Step 1. Assembly motor to base

Step 2. Fasten feet to base

- Ignore that there's a whole robot arm attached in the video.

Step 3. Attach dial to motor

Step 4. Connect and calibrate electronics

ELECTRONICS SAFETY: Make sure to separate the PCB from the metal base before turning on the power, otherwise the circuit will short! Either elevate the PCB above the base with the screws provided, or place the PCB next to the metal base on the table.

1. Turn on the system: press the power button on the PCB shield.
2. Calibrate: Press and hold the button on the C610 motor controller until the motor starts moving and release.
3. Wait until the C610 motor controller restarts.
4. Set ID: Click the button on the C610 controller, then a little while later (half second or so) press the button again. The light should flash green.
5. The light should now flash once every 2 seconds or so. The number of blinks indicates which ID it is. For example two blinks every 2 seconds indicates ID=2.

Important: To set a motor controller to a certain ID, click (short press) press to put the motor controller into id-setting mode, then click N more times in quick succession, where N is the desired ID. Eg, for a desired ID of 3, press 3 more times after the first click.

Step 4. Run the starter code

1. Git clone the [starter code](#), open in VSCode, and upload to Teensy.
1. Examine where in the code the motor angle and velocity are read in `src/main.cpp`. Examine where the motor is commanded.
2. Upload starter code to Teensy (right arrow icon in blue bar of VSCode or click the ant icon, then upload)
3. Open the serial monitor in VSCode (icon that looks like a plug in bottom bar of VSCode or click ant icon, then monitor)
4. Click into the serial monitor area and then press the key `s` to make the Teensy start printing out the angle and velocity of the connected motor.
5. Press `s` again to stop the program. If you want to rerun the code, upload again or unplug and replug your computer from the Teensy.

```
m0_pos: 0.00    m0_vel: 0.00    m1_pos: 0.00    m1_vel: 0.00    m2_pos: 0.00    m2_vel: 0.00
m0_pos: 0.00    m0_vel: 0.00    m1_pos: 0.00    m1_vel: 0.00    m2_pos: 0.00    m2_vel: 0.00
m0_pos: 0.00    m0_vel: 0.00    m1_pos: 0.00    m1_vel: 0.00    m2_pos: 0.00    m2_vel: 0.00
m0_pos: 0.00    m0_vel: 0.00    m1_pos: 0.00    m1_vel: 0.00    m2_pos: 0.00    m2_vel: 0.00
```

Fig. 3: Example output from serial monitor.

Step 5. Run bang-bang control

1. Uncomment the bang-bang code in `src/main.cpp` and upload.
2. Observe the effects of changing the current command to something else.
3. *FEEL* how the controller behaves. Move the dial by hand and see how the controller reacts.

Example bang-bang control.

Step 6. Write PD position control

1. Comment out the bang-bang controller.
2. Complete the `pd_control` function in `src/main.cpp`. Your function should return a current command (100mA, 200mA etc) using the PD control law $\tau = K_p * (\text{target} - \text{theta}) + K_d * (-\omega)$.
3. Use $K_p = 1000.0$ and $K_d = 0.0$ to start. Don't forget the negative signs!
4. Upload code to Teensy
5. *FEEL* the effect of the PD controller.
6. What happens when you rotate the disc just a little bit away from the target position? What happens when you rotate it a lot away from the target position? Do you feel the motor torque increase and then flatten out as you rotate the disc?

[Insert gif of proper PD joint control]

Step 7. Experiment with different parameters

Note: Some of these steps will cause the output disc to go unstable and violently shake, be prepared!

For each of these situations (except the ones that go unstable), rotate the disc around with your hand to get a physical sense for the PD behavior.

1. Keeping K_d constant (0), experiment with $K_p = -100$ and $K_p = 5000$. Discuss with your partner how each feels. How are K_p and stiffness related?
2. Keeping K_p constant (1000), experiment with different K_d values from -10 to 1000
3. See what happens when K_p is too high. Try $K_p=50000$ and $K_d=100$.
4. See what happens when K_d is too high. Try $K_p=0$ and $K_d=100000$.
5. See what happens with just moderate damping. Try $K_p=0$ and $K_d=100$.

The expected behavior is that higher K_p values will make the position control more stiff while higher K_d values will make the motor slower to achieve the desired position. If either gain is too high or is negative, the motor will go unstable.

[Insert gif of some instability]

Step 8. Experiment with different loop rates

1. Examine where the code is checking if it's time to issue another control update.
2. Change the update rate to 4Hz with $K_p=1000$ and $K_d=100$ to observe instability.

Step 9. Program periodic motion

1. Set the update rate back to 200Hz (5ms interval).
2. Program the motor to track a sinusoidal position, like the psuedocode below.

```
float time = millis() / 1000.0
position_target = sin(time)
```

3. Play around with different frequencies. How high can you raise the frequency before the motor no longer moves as much as you expect?

Fun fact, the maximum frequency you can go before the motor moves to only 71% (-3dB) of the intended motion is called the bandwidth.

[Insert gif of sinusoidal motion]

1.4.3 (Old) Mini-lecture - Joint Control

1.5 Lab 2 - Bad Robot Surgeon

Contents

- [Lab 2 - Bad Robot Surgeon](#)
 - [Lab Instructions](#)

1.5.1 Lab Instructions

Goal: Build two robot arms that mirror each other's motion.

Step 1. Connect 2 more motors

1. Connect power and encoder cables from motors to controllers.
2. Connect power and CAN cables from controllers to Pupper PCB. Make sure the CAN cables go into the same row (row near the Teensy).
3. Set the new motor controllers to have different IDs. Use 1, 2, and 3. **Important:** *To set a motor controller to a certain ID, click (short press) press to put the motor controller into id-setting mode, then click N more times in quick succession, where N is the desired ID. Eg, for a desired ID of 3, press 3 more times after the first click. Unplug the CAN (small) cables from the PCB while setting ID's to make it easier, otherwise they may conflict while you are changing them*

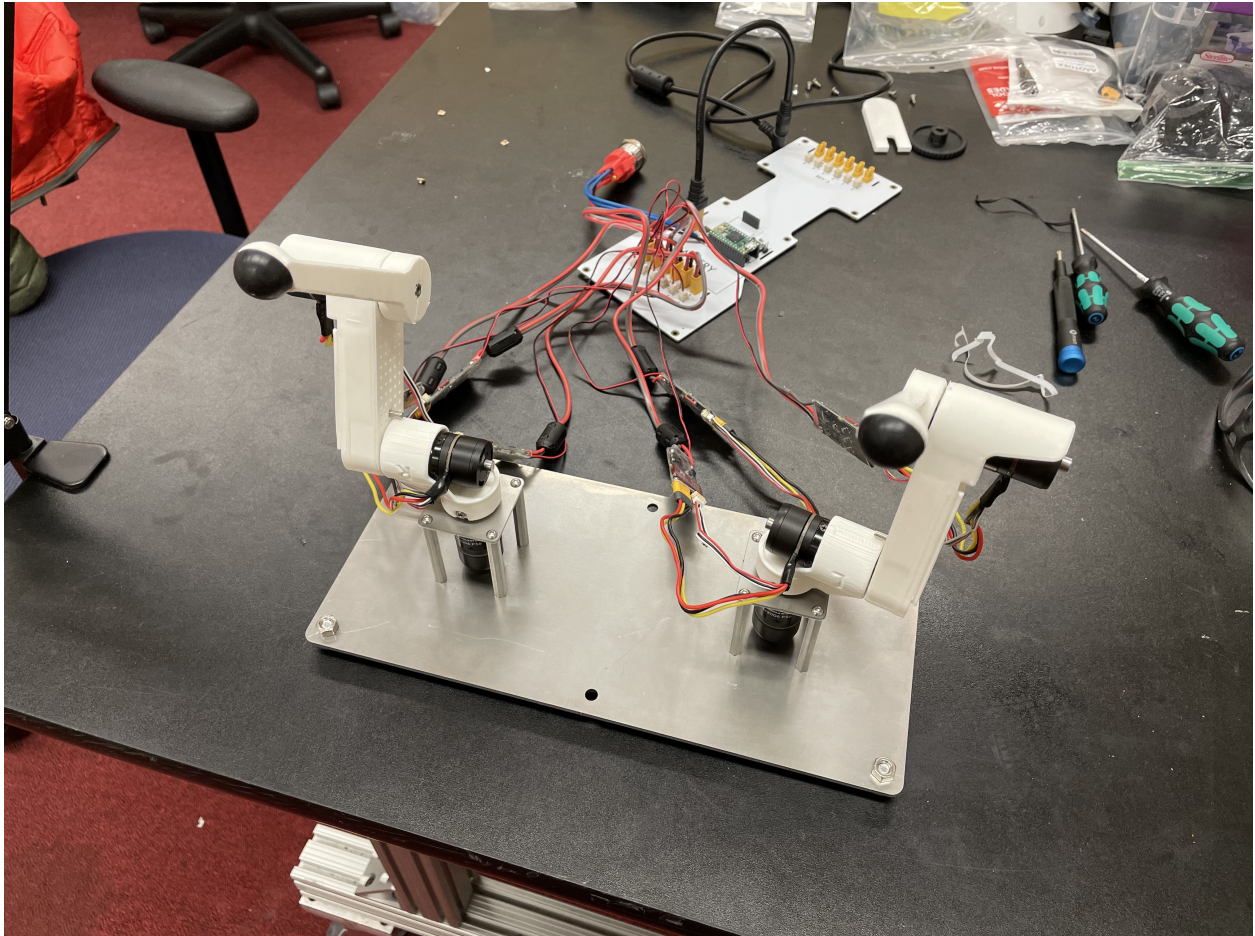


Fig. 4: Assembled teleop robot arms

Step 1.5. Control the 3 motors together

1. Modify your PD control code from lab 1 to control all 3 motors (controlled independently).

[TODO: insert pic of completed setup]

Step 2. Insert square nuts into plastic parts

You may need to use force to get them in.

Step 3. Connect and calibrate electronics

IMPORTANT: Calibrate each motor *before* you assemble the arm, so that they get an accurate calibration with no load. Once you calibrate the motor, keep it with the motor controller you calibrated it on. The calibration is dependent on the motor controller, so that motor is now paired with the motor controller. If in doubt, recalibrate.

Step 4. Assemble the three motors into a robot arm!

IMPORTANT: Make sure you calibrated the motors before assembling the arm!

The robot arm we're making is actually one of Pupper's right legs so you'll see the instructional videos reference it as such.

Assembly Pro Tips

1. Use the tip of the hex driver to align the shoulder bolt with the square nut. This will make assembly much easier.
2. Use force when inserting the shoulder bolts. Sometimes it is hard to get them through the hole in the motor shaft even if perfectly aligned.
3. Tighten the shoulder bolts as tight as possible without stripping. It is vital for the future stability of your Pupper!

Step 5. Run your code again on the new robot arm

1. Note that the “zero” position of these motors is whatever position it was at when the Teensy and motor were first both powered on.
2. Upload and run your code for controlling the 3 motors simultaneously.

Example where the arm PID positions targets are set so that it stands up vertically.

Step 6. Connect three more motors to use as control dials

1. Connect three additional motors to the same CAN bus (ie same row of connectors).
2. Calibrate and connect three additional motors to the Pupper PCB.
3. Set their IDs to not overlap with your existing motors. We use 4, 5, and 6.
4. Set the target positions of the base motor, shoulder motor, and elbow motor to the angle readings of the first, second, and third new motors respectively.

[TODO: gif]

Step 7. Connect and calibrate electronics for second robot arm

Step 8. Assemble the three new motors into a robot arm

We’re now making one of Pupper’s left-side legs to use as the second robot arm.

Step 9. Use the arms as leader and follower.

1. Use the same code as in Step 6 where one set of motors controllers the other.
2. Start the robot arms from the same position.
3. Tune Kp and Kd gains and maximum current as you like.

[TODO: pic]

Step 10. Make the robot arms bidirectional!

1. Program position control for the leader arm actuators (formerly control dial actuators)
2. Set the position targets of the leader arm to the positions of the follower arm.
3. Assuming the leader arm has controller IDs 1, 2 and 3, and the follower arm has controller IDs 4, 5 and 6, you can send current (ie torque) commands to the robot arms with the code

```
bus.CommandTorques(m0_current, m1_current, m2_current, m3_current,
↳C610Subbus::kOneToFourBlinks);
bus.CommandTorques(m4_current, m5_current, 0, 0, C610Subbus::kFiveToEightBlinks);
```

4. Congrats. Play with your robot! Make modifications!

[TODO: gif]

1.6 Lab 3 - Forward Kinematics**Contents**

- *Lab 3 - Forward Kinematics*
 - *Lecture*
 - *Lab Instructions*

1.6.1 Lecture

Forward kinematics lecture

Follow-up lecture to clarify some things

1.6.2 Lab Instructions

Goal: 1) Learn how to compute forward kinematics 2) Become familiar with simulation-to-real pipeline.

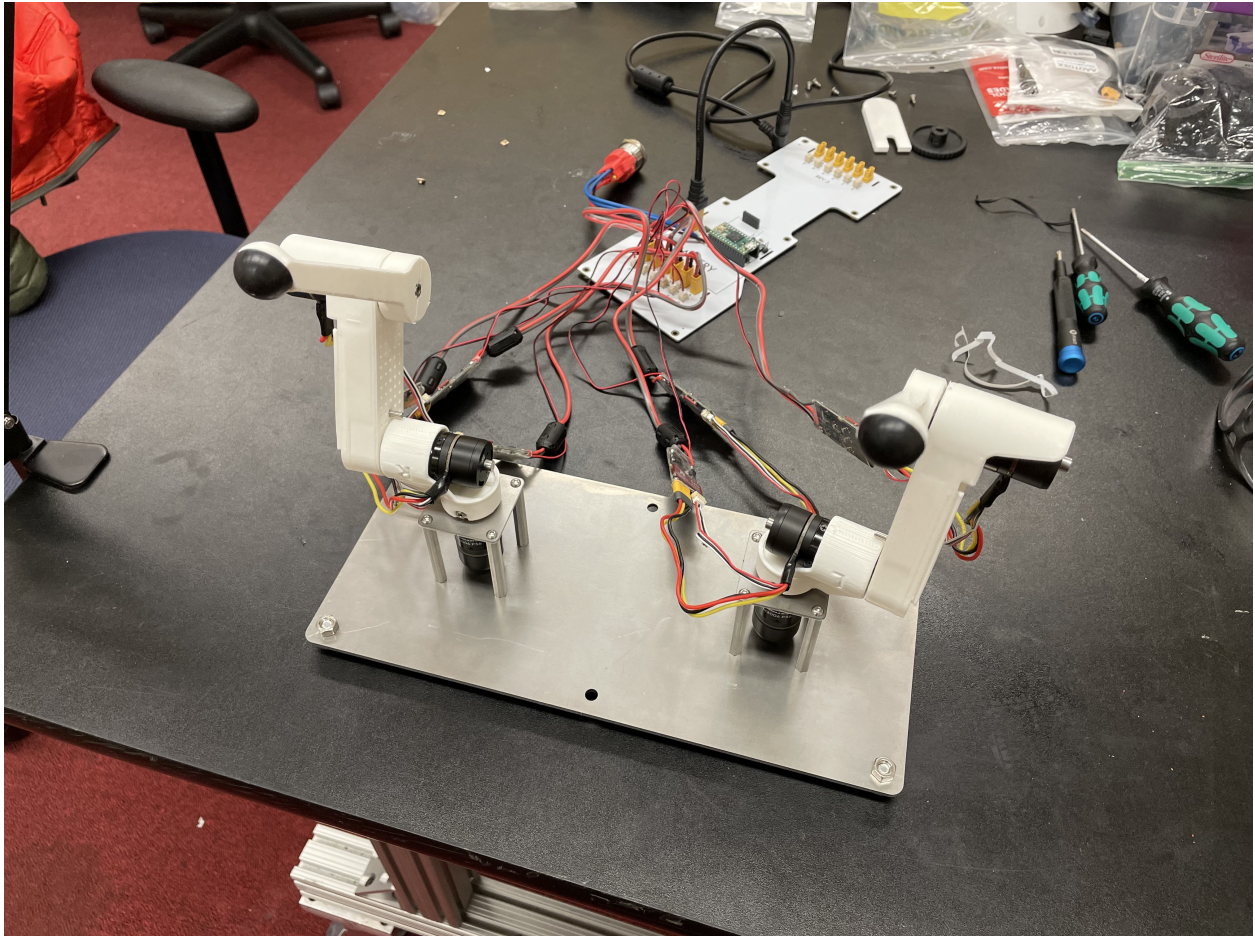


Fig. 5: TODO: picture of sim and real robot together

Step 0. Flash Teensy with our firmware

1. Download <https://github.com/stanfordroboticsclub/reacher-lab/blob/main/firmware.hex>
2. Follow the instructions in this video to flash the Teensy. In the video when they say to use a paperclip to press the button, you can just press the button on the Teensy directly instead. https://www.youtube.com/watch?v=9PyiGUO9_KQ. If you can't find the Teensyloader application on your computer, you can upload any random Teensy code through VSCode and that will open the Teensyloader window which you can then use to flash the Teensy.

Step 1. Install simulator

Follow the instructions available at <https://github.com/stanfordroboticsclub/reacher-lab> to install and run the reacher code in simulation

Step 2. Test simulator

Run `python reacher/reacher_manual_control.py`

Step 3. Test robot simultaneously with simulator

Run `python reacher/reacher_manual_control.py --run_on_robot`

The robot should match the motion of the robot in the simulator. Note: if you click and drag the robot in the simulator, the real robot will NOT follow. The real robot is following the joint angles set by the sliders.

Step 4. Code forward kinematics

1. Complete the function `calculate_forward_kinematics_robot` in `reacher/reacher_kinematics.py`.
2. Run `python reacher/reacher_manual_control.py` to verify with the white ball. Ensure you are in the correct directory

The output of your kinematics functions sets the white ball's position in the simulator, so ideally, the red ball fixed to the end of the robot arm should track the white ball nearly perfectly.

The result should look something like this

Debugging tips

1. Use numpy operators for your matrix operations
2. Check the shape of your vectors and matrices matches what you expect. This is where most students run into trouble. Remember the rules of matrix multiplication.

1.7 Lab 4 - Inverse Kinematics

Contents

- *Lab 4 - Inverse Kinematics*
 - *Inverse Kinematics Lecture*
 - *Lab Instructions*
 - *(Old) IK Lecture*

1.7.1 Inverse Kinematics Lecture

Inverse Kinematics lecture by Jaden Clark, April 24, 2023.

1.7.2 Lab Instructions

Goal: 1) Learn how to compute inverse kinematics 2) Become familiar with simulation-to-real pipeline.

Step 1. Code inverse kinematics

1. Implement `ik_cost` in `reacher_kinematics.py` as the squared-norm of the error between the position returned by `FK(guess)` and `end_effector_pos`.
2. Implement `calculate_jacobian` in `reacher_kinematics.py` using the finite differencing method we covered in lecture. A good perturbation size is small, like 0.001.
3. Implement `calculate_inverse_kinematics` in `reacher_kinematics.py`. Play around with different step sizes, from 1 to 100, to see what works for you.

Step 2. Test the consistency between forward kinematics and inverse kinematics

1. Test your code by taking some reachable (x,y,z) point in space (think hard about what's reachable!), using your IK function to get the corresponding joint angles, then passing them to your FK function to retrieve the original (x,y,z). These should match as long as the original point was reachable.

The reason we're doing this IK -> FK consistency test and not a FK -> IK consistency test is that for any reachable point in space, the robot can flip its "elbow" joint up or down to get to that point in space, resulting in different joint angles.

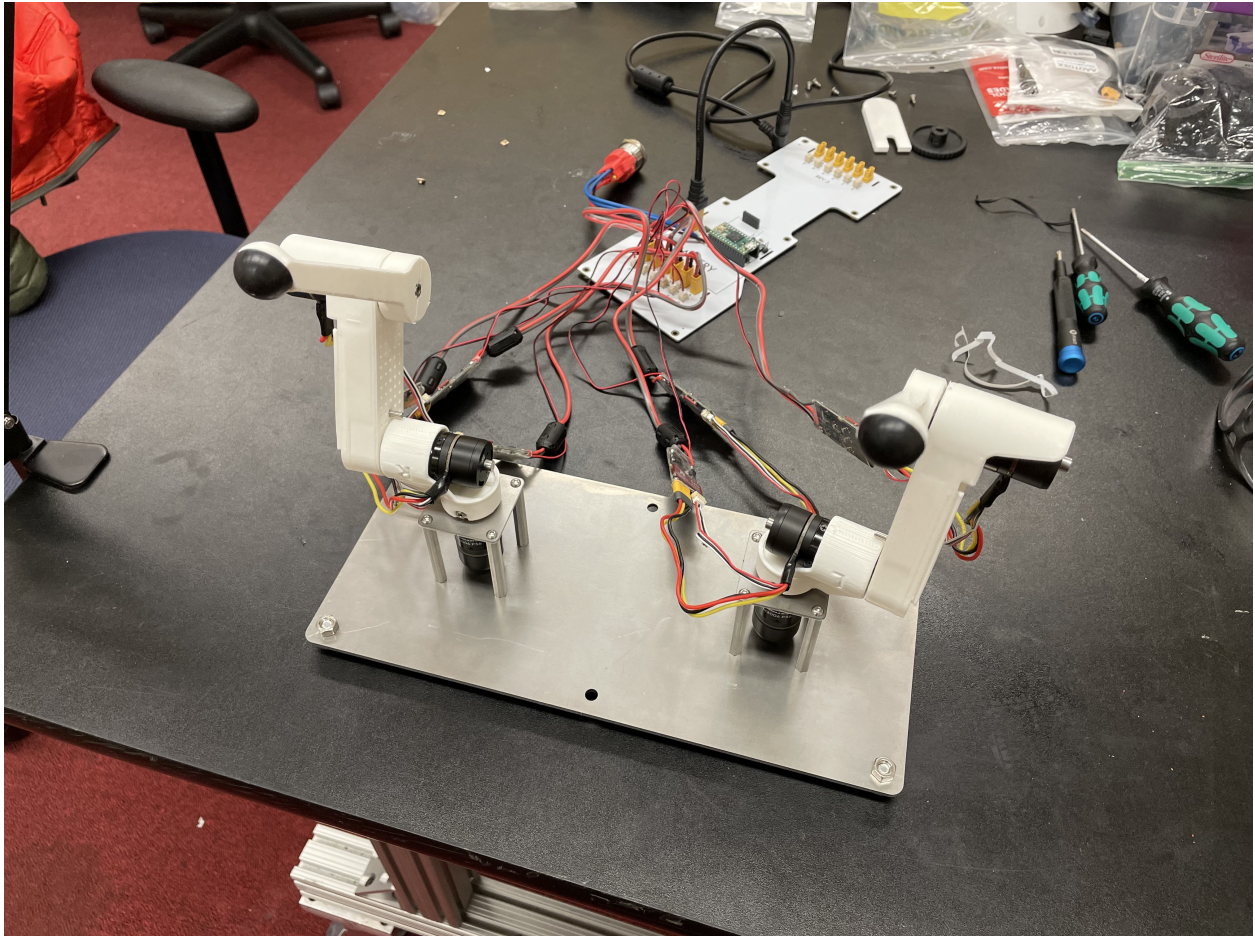


Fig. 6: TODO: picture of sim and real robot together

Step 3. Test IK in simulation

1. Use the simulator to visualize your IK: Make the white sphere go to an arbitrary point in space by modifying `reacher_manual_control.py` (see line 95)
2. Use your IK code to figure out the joint angles, and then set the simulated robot's joint commands to those angles (see line 81).
3. Like in lab 3, the white sphere should follow the red sphere of the robot.

Step 4. Deploy your IK to your robot

1. Look at the code that sets the actual robot motors, specifically line 86 and 87.
2. Change the joint angles sent to the hardware to those calculated by your IK.
3. Deploy to robot. Remember to pass the `--run_on_robot` flag when you run your program.

Step 5. Make your two robot arms match each other's end-effector positions

1. Calculate the cartesian end-effector position of the right arm using FK.
2. Use this result to calculate the cartesian position of the right arm's end-effector relative to the base of the left arm.
3. Disable the right arm's torque by de-activating the motors in the right arm. [TODO link code line number]
4. Deploy to robot and sanity check the reported position
5. Figure out what to add/subtract from the right arm's position to get the corresponding position relative to the left arm.
6. Deploy to robot and sanity check that the position relative to the left arm is correct.
7. Use your IK to move the left arm to this position in the simulator. Check that the left arm doesn't freak out.
8. Deploy to robot!
9. The result you should get is that the left arm and right arm are touching or very close at the end effector! You can put an object between the end-effectors and now you have a multi-robot manipulator!

Notes: * The code is set up to control both arms, but currently we are commanding the second robot arm's actuators to stay at zero. * We can move the second arm by setting the values in the 3rd column (index 2) of the `full_actions` 2d array. See line 86 as an example. * You might need to change the sign of some of your joint angle commands because the second arm is a right-sided leg instead of a left-sided leg.

1.7.3 (Old) IK Lecture

1.8 Lab 5 - Reinforcement Learning

Contents

- *Lab 5 - Reinforcement Learning*
 - *Lecture 5.1: Intro to RL and Its Applications in Robotics*
 - *Lecture 5.2: Reinforcement Learning on Legged Robots*
 - *Lecture 5.3: Embodied Intelligence*
 - *Lab instructions*
 - *(Old) RL Lecture*

1.8.1 Lecture 5.1: Intro to RL and Its Applications in Robotics

Intro to RL and Its Applications in Robotics Lecture by Tingnan Zhang, May 3, 2023.

1.8.2 Lecture 5.2: Reinforcement Learning on Legged Robots

Reinforcement learning on legged robots lecture by Jie Tan, May 10, 2023.

1.8.3 Lecture 5.3: Embodied Intelligence

Embodied Intelligence lecture by Adrien Gaidon, May 15, 2023.

Zoom password (if prompted): ^dgGqIB5

1.8.4 Lab instructions

These instructions assume you are running Mac or Linux. If you have Windows 10 or lower, I recommend dual-booting linux. If you have Windows 11, try using the Windows Linux Subsystem. Otherwise proceed at your own risk!

Step 1. Set up simulation environment

1. Clone the simulator repository `git clone https://github.com/jietan/puppersim.git`
2. Follow the instructions in the [System setup](#) and [Getting the code ready](#) sections of the Puppertime README.md.

Step 2. Train the policy using RL (ARS)

1. Follow instructions at <https://github.com/Nate711/puppersim/blob/main/puppersim/reacher/README.md> to run the commands to train the policy.
2. Wait about 50 iterations until going to step 3 but leave it training

Step 3. Run your policy in simulator

1. Follow instructions at <https://github.com/Nate711/puppersim/blob/main/puppersim/reacher/README.md> to run the policy.

Step 4. Deploy policy to robot

1. Follow instructions at <https://github.com/Nate711/puppersim/blob/main/puppersim/reacher/README.md> to deploy to your robot.

Step 5. One day project of your choice

Do your own mini project!

Some ideas:

- Teach the robot to trace out a specific shape in the air. (medium)
- Teach the robot to turn itself off by pressing its power button. (medium)
- Add a cube in the pybullet simulation and teach the robot to kick it. (hard)
- Turn off torque on the elbow or shoulder motor and make the robot learn to balance the arm vertically. (hard)

1.8.5 (Old) RL Lecture

<https://share.icloud.com/photos/0836FiHhLJuCXC9TyqSW8Ilw>

1.9 Lab 6 - Pupper Assembly

Contents

- *Lab 6 - Pupper Assembly*
 - *Lab instructions*
 - *Resources*

1.9.1 Lab instructions

Step 1. Leg Assembly

- Make 1 more left leg and 1 more right leg using instructions in lab 2, however, do not attach the “base” motor

Step 2. Bulkhead assembly

1. Detach your robot arms from the base
2. Attach 2 motors, one into each side of motor bulkhead, with 3 M3x6 phillips head screws each
3. Attach left and right legs to the shafts of the motors you just installed and screw in shoulder bolt tightly
4. Thread the wrapped cables through slots
5. Zip tie the cables to keep them in place
6. Clip motor controllers into place
7. Connect motor and encoder cables to motor controllers
8. Repeat this process so you have two motor bulkheads with four legs total

CHECK. Ensure Motor IDs are correct

Use this diagram to ensure you have the correct IDs set on your motor controllers. For each bulkhead you should have:

Right left/right motion hip motor: 1

Right forward/back motion hip motor: 2

Right knee motor: 3

Left left/right motion hip motor: 4

Left forward/back motion hip motor: 5

Left knee motor: 6

Motor IDs

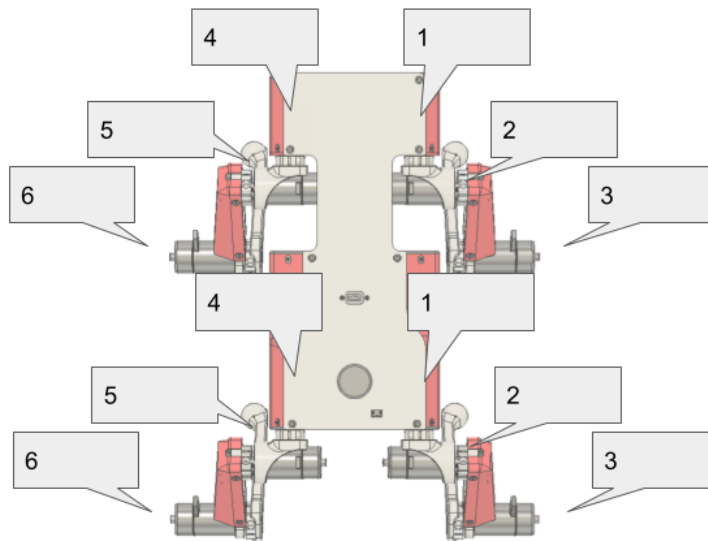


Fig. 7: Motor ID diagram

Step 3. Body assembly

1. Place front motor bulkhead
2. Connect motor controller power cables (yellow XT30) and CAN connectors (small white JST GH) to bottom PCB
3. Place back motor bulkhead and connect cables
4. Flip robot and fasten bulkheads to bottom PCB with 4x M3x6 button head screws
5. Tighten these screws well and/or add loctite

Step 4. Electronics bulkhead assembly

1. Screw RPi into electronics bulkhead with M2.5x5 socket head screws such that the Pi is oriented like in the video.
2. Connect USB C extension cable to Rpi
3. Connect RPi camera flex cable into RPi. There's a little grey flap that flips up on the connector that lets you slide the cable in. Flip the flap down to lock the cable in.
4. Connect RPi to power by using 2-pin cable. Connect one end into 5V, GND pins near the Teensy and other side into RPi. Quadruple-check that the 5V and GND pins are going the right places. See diagram.
5. Connect RPi to Teensy using USB A to USB micro cable

6. Connect RC receiver to RPi with usb extension cable.

Step 5. Top panel assembly

1. Insert the XT60 female side (conductor is a circular slot) of XT60 splitter cable into 3D printed power hub.
2. Insert JST-XH extender balance cable into 3D printed power hub.
3. Attach the 3D printed power hub to the top PCB with 2 M3x6 button head screws.
4. Take the large nut off the power switch and then mount the power switch to the top PCB panel. Then secure the switch by threading on the nut from the bottom of the top panel.
5. Screw the USB-C connector to the top PCB with 2 M3x6 button head screws
6. Connect other female XT60 into the bottom PCB

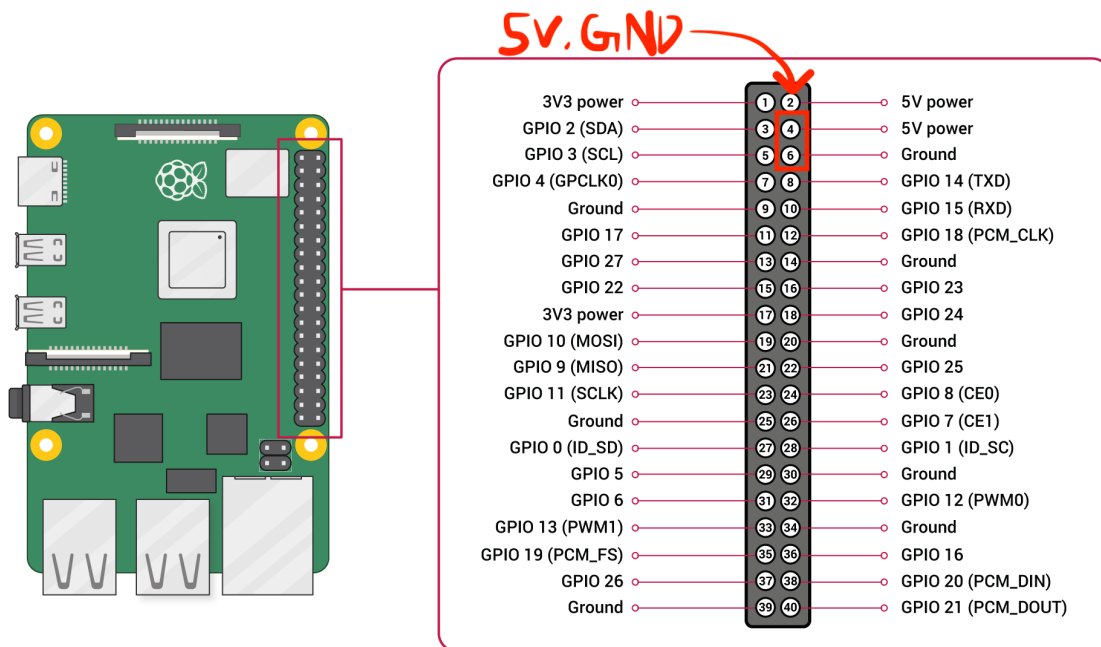


Fig. 8: RPi pinout.

Step 6. Attach front panel

1. Attach the Pi Camera to the front Pupper panel with M2x4 socket head cap screws
2. Attach the front Pupper panel with 2 M3x6 button head screws to the bottom PCB



Fig. 9: Bulkhead wiring.

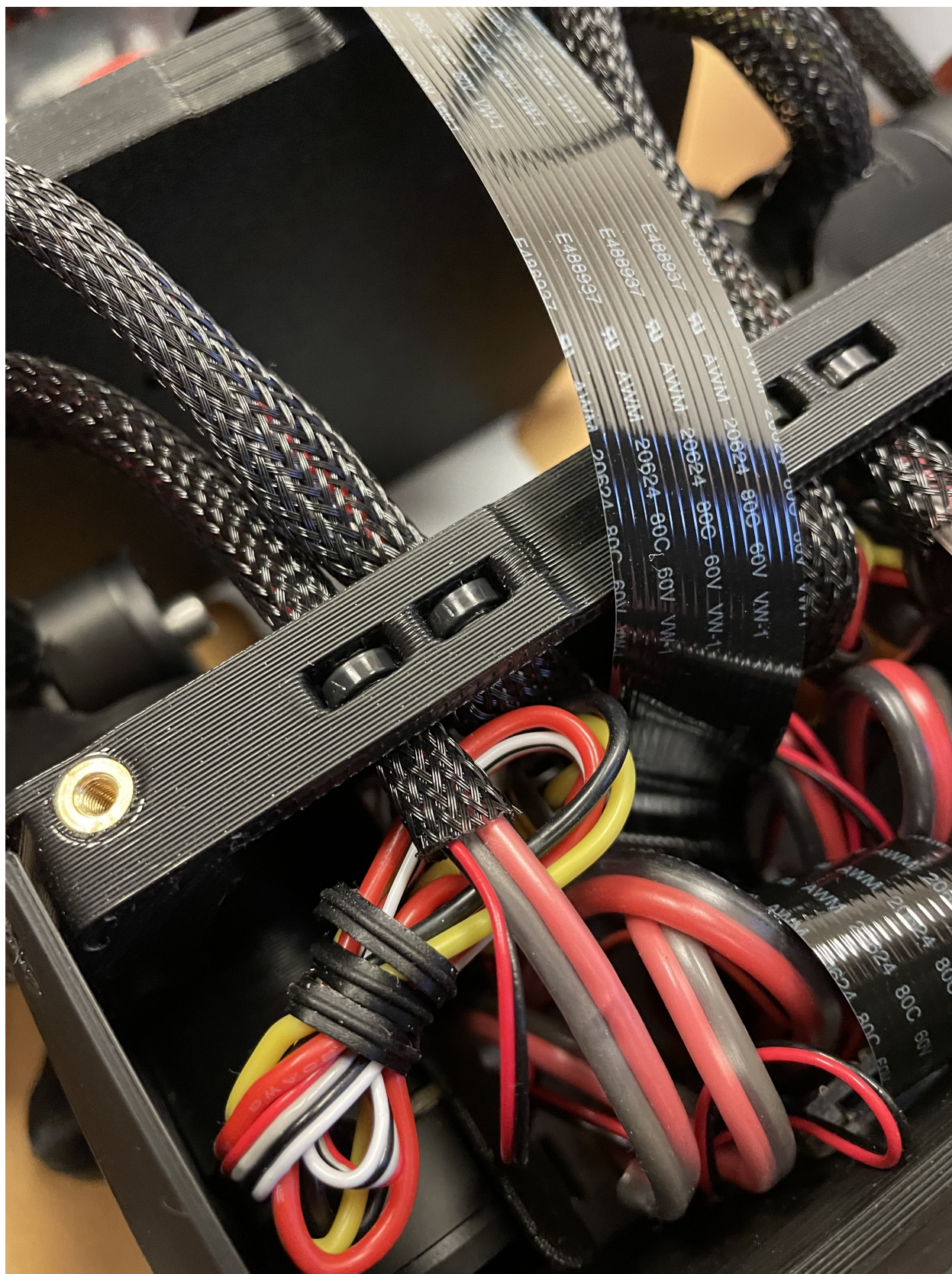


Fig. 10: Zip-tie close up.

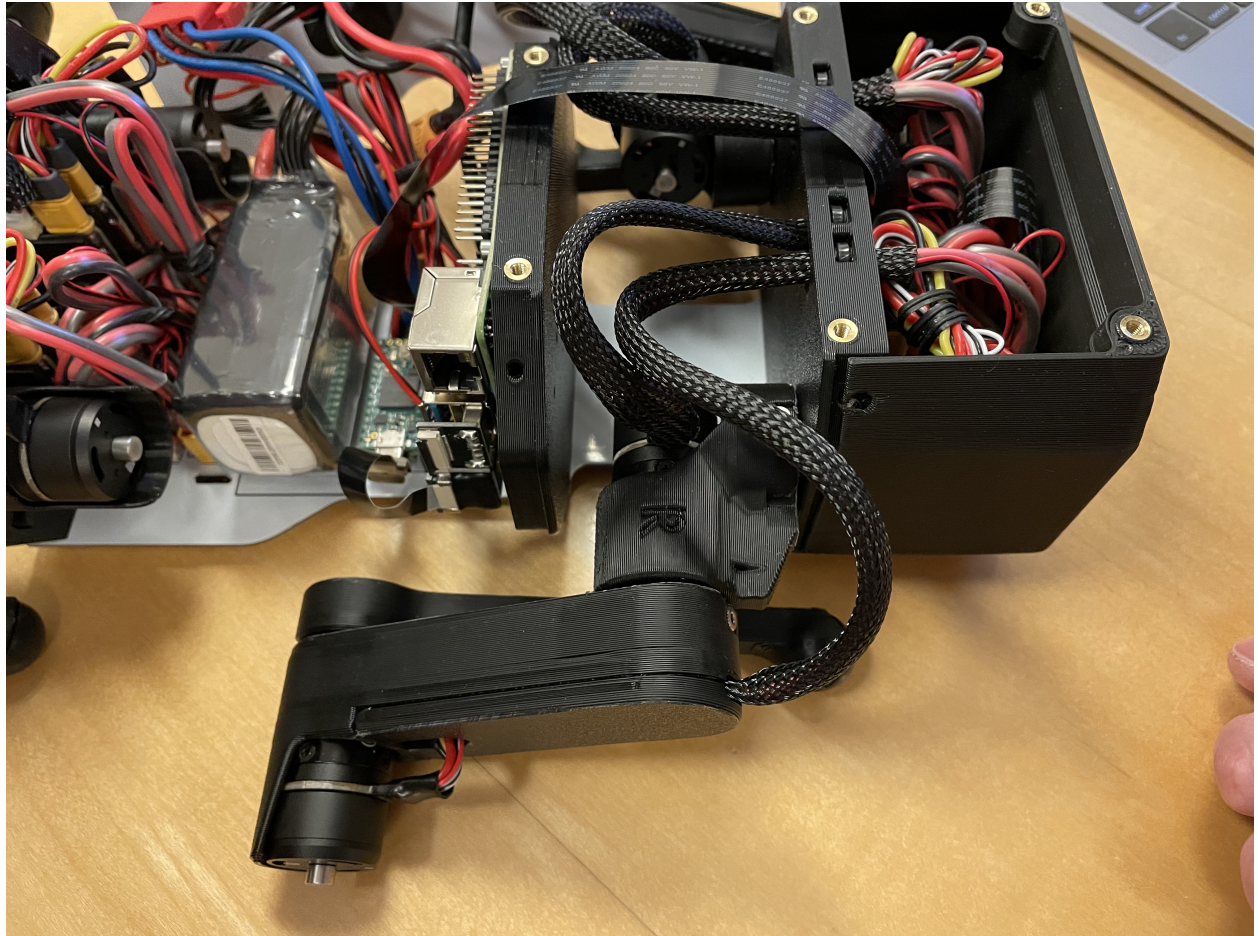


Fig. 11: Leg assembly.

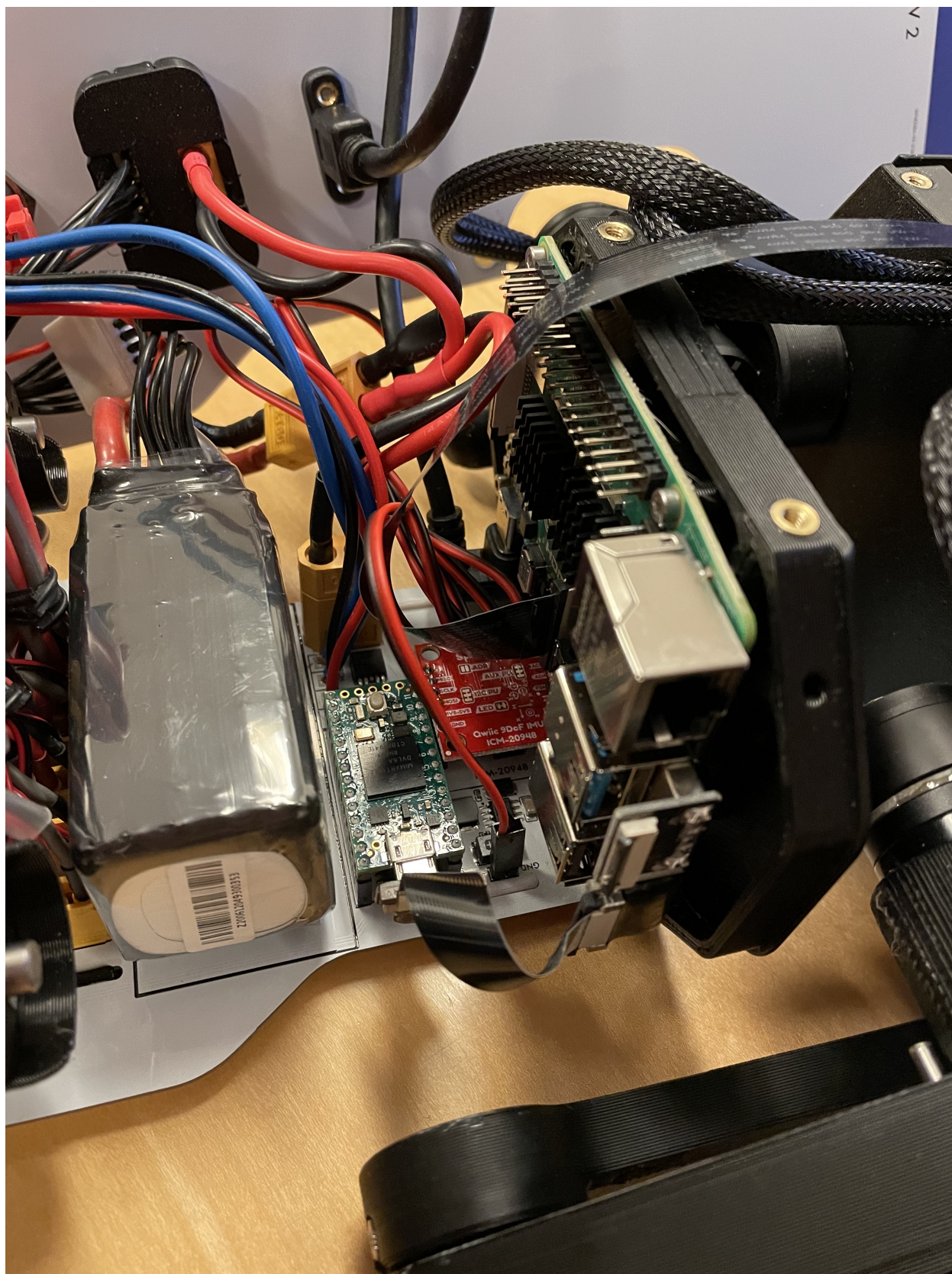


Fig. 12: Electronics assembly.



Fig. 13: Bottom view of top PCB.



Fig. 14: Top view of top PCB.

Step 7. Bind RC receiver

1. Get a FRSKY USB receiver and a BetaFPV Transmitter
2. While holding the button on the USB receiver down, insert it into your computer. It should show a flashing red light.
3. Then turn on the transmitter: Press and hold the power button for about 5 seconds until twice vibration. The LED will be green first. Wiggle the left stick until the LED turns blue. Then the LiteRadio is powered on.
4. Press the BIND button from the back of the transmitter. The transmitter will enter the binding mode and last about 10 seconds, indicated by blue and red LED flash alternately.
5. Once bound, the receiver should then show a solid (not blinking) green color.
6. Unplug and re-plug the receiver into your computer and restart the transmitter. Then go to <https://gamepad-tester.com/> in CHROME (no other browsers will work) to test that the receiver is receiving messages from the transmitter.
7. Refer to the transmitter manual for more info if needed <https://support.betafpv.com/hc/en-us/articles/900003583046-Manual-for-LiteRadio-2>.
8. Refer to the receiver manual for more info if needed <https://www.frsky-rc.com/wp-content/uploads/Downloads/Manual/XSR-SIM/XSR-SIM-%20manual.pdf>

Step 8. Finish hardware assembly

1. Put velcro or dual-lock onto the bottom PCB where it says “battery”. For now we’ll use the power supply to run the robot so you don’t have to install the actual battery.
2. Attach the top PCB panel with M3x6 button head screws.
3. Check again with instructors.
4. Marvel at your work!

Step 9. Flash code onto the Teensy

1. Go to <https://github.com/Nate711/DJIPupperTests/blob/master/README.md> for instructions on how to download and set up the Teensy firmware
2. Use VSCode PlatformIO to open the DJIPupperTests folder as a project and then upload the code to the Teensy. (Same thing as in labs 1-4).

Step 10. Flash software image onto Raspberry Pi

1. Download our [image](#)
2. Install [Balena etcher](#)
3. Flash the image onto the micro SD card using Balena etcher.
4. Insert the micro sd card into the Pi’s micro sd card slot (on bottom side of board)

Step 11. Enable the heuristic controller

1. Connect the robot to your computer via the top USB-C port on the robot.
2. SSH into the robot with `ssh pi@raspberrypi.local`. The password is `raspberry`. Ask for help if this doesn't work.
3. Run `sudo systemctl enable --now robot` to turn on the heuristic controller.
4. Run `sudo systemctl status robot` to check that the service is running ok (should be green).
5. Reboot with `sudo reboot 0`

Step 11.5. (Optional, Stanford only) Get Stanford Wifi access

1. Once ssh'd into the robot, run `ifconfig` and record the `wlan0` MAC address. This is the MAC address for the WIFI chip. It should be a series of hex characters like `f0:2f:4b:07:ee:ea`.
2. Go to `iprequest.stanford.edu` on your computer, and make a new registration for the Pi. Select other device -> other wired. Enter the *Wireless* MAC address you got in step 1.
3. Restart the Pi and SSH back in
4. Run `sudo raspi-config`, go to System options -> Wireless LAN.
5. Enter Stanford as the network name and leave password empty
6. You might have to restart the Pi a few times, and use `raspi-config` to set the desired network a few times for it to work.
7. To test if the Pi now has internet access, run `ping www.google.com`. It should say you're getting bytes back from Google.
8. If it doesn't work, wait 20 minutes, restart the Pi, and try again!
9. Make sure you change the password after connecting to Wifi or it *will* get hacked. We highly recommend physically labeling the robot with the new password.

Step 12. Take your robot on a walk

1. Unplug the Pi from USB C.
2. Place your robot on a flat, level surface. Position the legs as shown in the picture below.
3. Power on the robot by hooking up the power supply to the bottom PCB (like you've done in labs).
4. Connect the Pi with USB C to your computer.
5. Wait for the robot to complete the calibration sequence. During the calibration sequence, the hips should turn inwards until they hit the stops, then back down. Then the thigh pieces should rotate upward until they hit their stops and then back down. **TODO** Add calibration video
6. Flip all switches on the back RC transmitter down so they're away from you.
7. Turn on the RC transmitter by pressing the middle power button and moving the left joystick up and down until the light turns blue.
8. Wait ~30s for the RPi to boot (the green light should stop blinking).
9. Flip the lower left switch on the controller up to enable the robot. It'll move!
10. Flip the lower right switch on the controller up to start the robot trotting.

11. Enjoy your hard work and play with Pupper!

- The top right switch flips between trotting and walking.
- Left/right on the left joystick controls turning.
- Up/down on the right joystick controls forward/back.
- Left/right on the right joystick controls strafing left/right.



Fig. 15: Startup position.

1.9.2 Resources

Wiring diagram

1.10 Lab 7 - Pupper Control and Simulation

Contents

- *Lab 7 - Pupper Control and Simulation*
 - *Mini-lecture - TBD*
 - *Lab instructions*

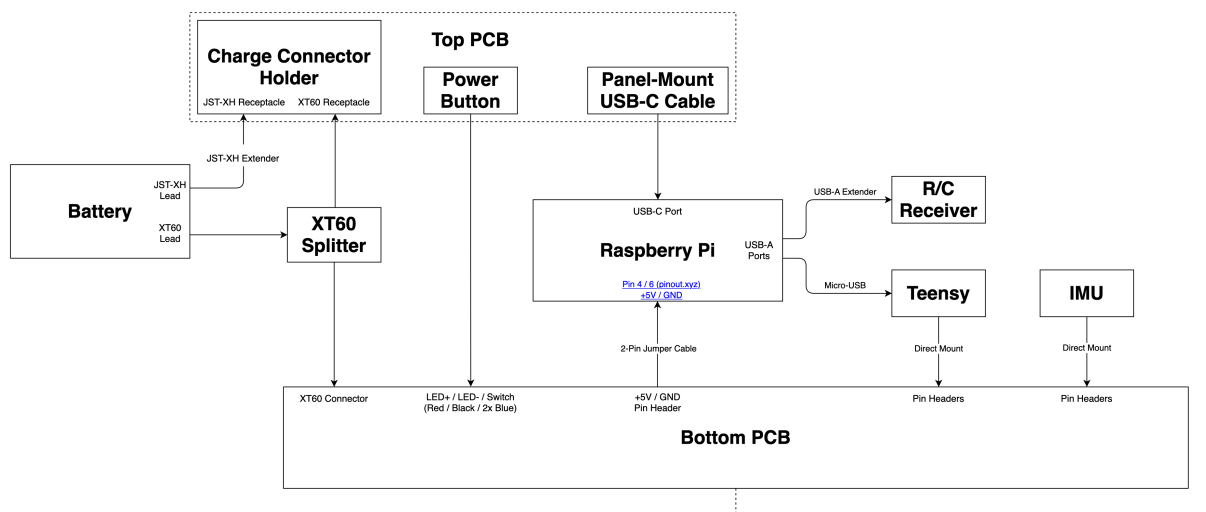


Fig. 16: Wiring diagram.

1.10.1 Mini-lecture - TBD

Slides: <https://docs.google.com/presentation/d/1yjNz9ybcmlJLKLkAnhHgILDR1hkjVbzhAphLXwZrO3mI/edit?usp=sharing> |

1.10.2 Lab instructions

These instructions assume you are running mac or linux. If you have Windows 10 or lower, I recommend dual-booting linux. If you have Windows 11, try using the Windows Linux Subsystem

Step 1. Set up simulation environment

1. `git pull` the latest `puppersim` repository, which is the same you used in Lab 5 <https://github.com/jietan/puppersim/>.
2. Install additional dependencies `pip3 install numpy transforms3d pyserial`. Use `pip` if `pip3` doesn't work.

Step 2. Try out the simulator

1. Follow the instructions in the **Simulating the heuristic controller** section of the puppersim repository.
2. **Check out these keyboard controls:**
 - wasd: → moves robot forward/back left/right.
 - arrow keys: → turns robot left/right
 - q: → activates robot
 - e: → starts trotting
 - ijkl: → tilts robot
3. To activate the robot, press q. To start trotting, press e.
4. Enjoy the simulator!
5. To close the simulator, do not press use the usual close button in the top left of the simulator window. Instead do Ctrl-C in both terminal tabs/windows.

Expected result:

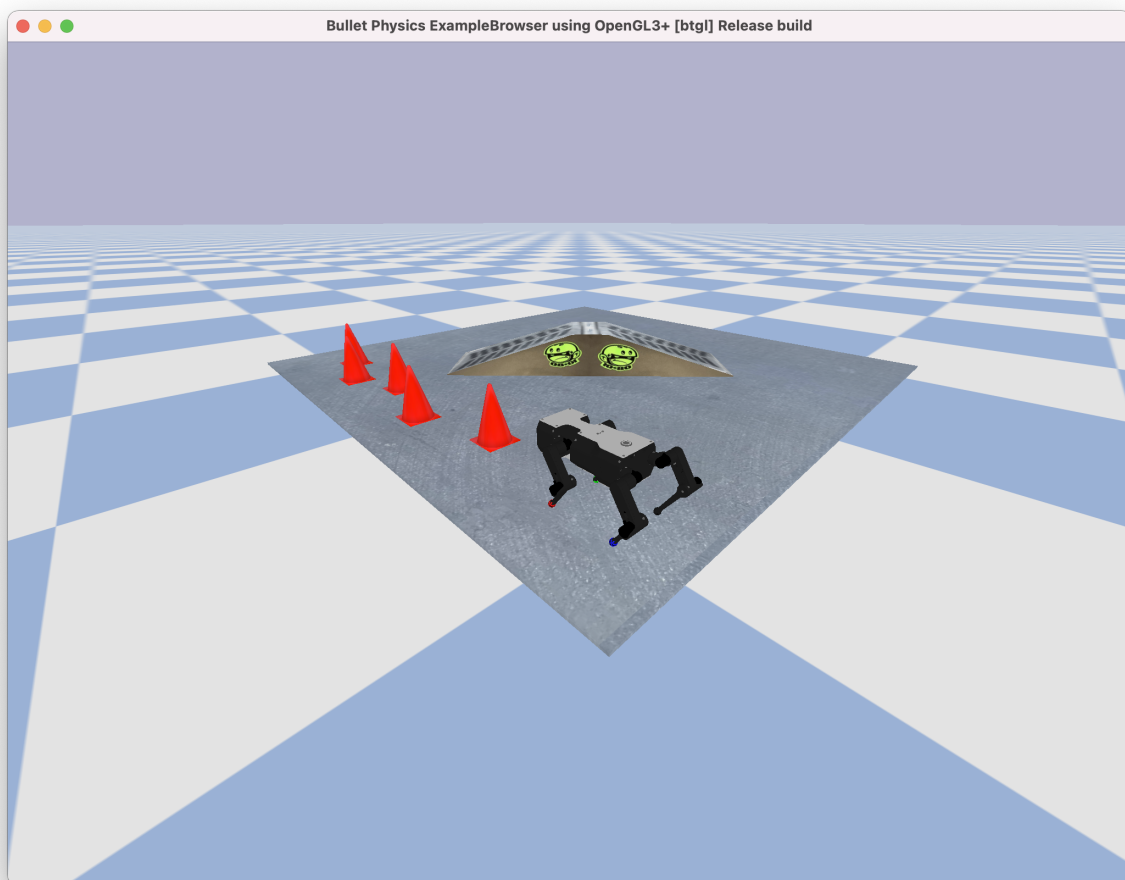
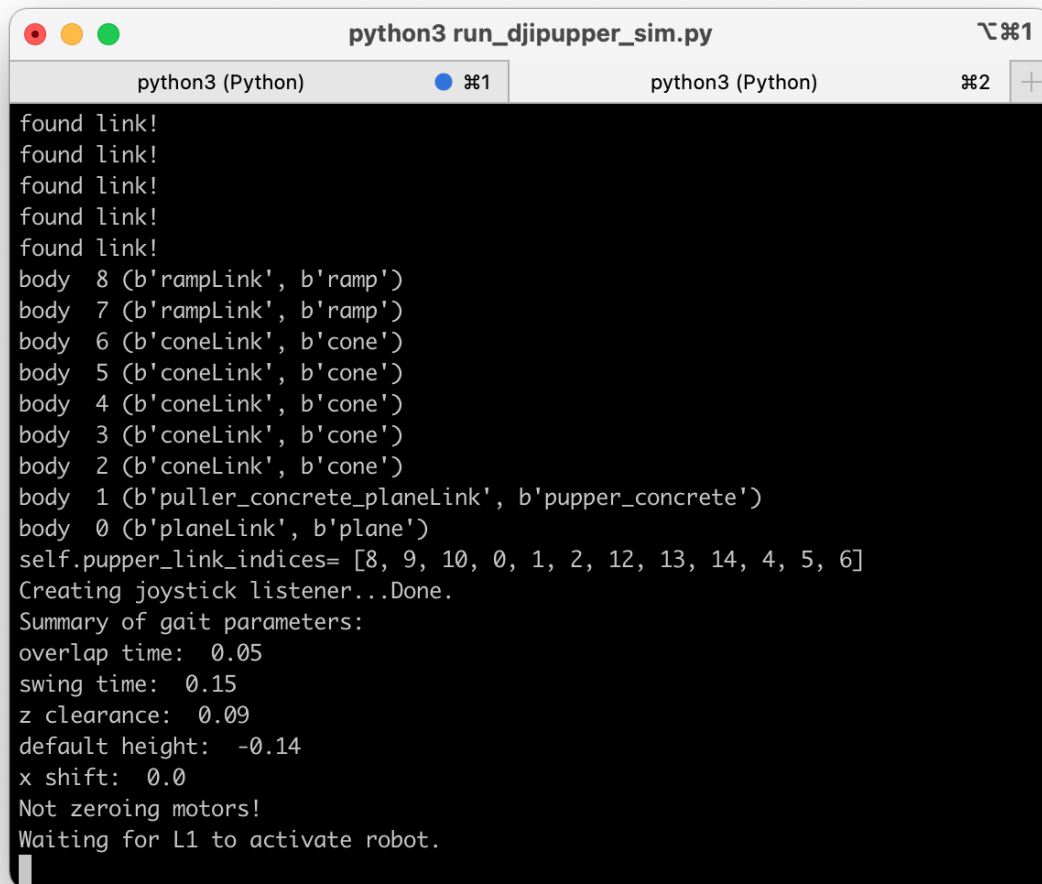


Fig. 17: Expected simulator window.



The image shows a screenshot of a macOS terminal window titled "python3 run_djipupper_sim.py". The window has two tabs, both labeled "python3 (Python)". The terminal output is as follows:

```
found link!
found link!
found link!
found link!
found link!
body 8 (b'rampLink', b'ramp')
body 7 (b'rampLink', b'ramp')
body 6 (b'coneLink', b'cone')
body 5 (b'coneLink', b'cone')
body 4 (b'coneLink', b'cone')
body 3 (b'coneLink', b'cone')
body 2 (b'coneLink', b'cone')
body 1 (b'puller_concrete_planeLink', b'pupper_concrete')
body 0 (b'planeLink', b'plane')
self.pupper_link_indices= [8, 9, 10, 0, 1, 2, 12, 13, 14, 4, 5, 6]
Creating joystick listener...Done.
Summary of gait parameters:
overlap time: 0.05
swing time: 0.15
z clearance: 0.09
default height: -0.14
x shift: 0.0
Not zeroing motors!
Waiting for L1 to activate robot.
```

Fig. 18: Expected output from `python3 run_djipupper_sim.py`.

Step 3. Experiment

1. Try changing `self.overlap_time` (line 52) in `StanfordQuadruped/djipupper/Config.py`. This controls the amount of time in which all four legs are on the ground for the trot.
2. Try changing `self.swing_time` (line 53) in `StanfordQuadruped/djipupper/Config.py`. This controls the amount of time in which just two legs are on the ground for the trot.
3. Mess around with other things:

Other available parameters

Maximum commands:

- `max_x_velocity`: Maximum forward velocity. Default: 0.6. Reasonable range: [0, 1.2]. Units: [meters/sec].
- `max_y_velocity`: Maximum horizontal velocities. Default: 0.6. Reasonable range: [0, 1.2]. Units: [meters/sec].
- `max_yaw_rate`: Maximum turning rate. Default: 2.5. Reasonable range: [0, 4]. Units: [rad/sec].

Foot placement parameters:

- `delta_x`: When the robot is trotting in place, this is 1/2 the forward/back distance between the feet. Default: 0.1. Reasonable range: [0.05, 0.15]. Units: [meters].
- `delta_y`: When the robot is trotting in place, this is 1/2 the left/right distance between the feet. Default: 0.1. Reasonable range: [0.05, 0.15]. Units: [meters].
- `x_shift`: This is the amount that the robot walks in front of it's center of mass vs behind. Default: 0.005. Reasonable range: [-0.03, 0.03]. Units: [meters].
- `default_z_ref`: Default amount that the feet are beneath the CoM of the robot. Default: -0.14. Reasonable range: [-0.05, -0.18]. Units: [meters].

Trotting parameters:

- `overlap_time`: 1/2 the duration of time in each trot cycle when all four feet are on the ground. Default: 0.05. Reasonable range: [0, 0.25]. Units: [seconds].
- `swing_time`: 1/2 the duration of time in each trot cycle when two feet are on the ground and two are in the air. Default: 0.15. Reasonable range: [0, 0.25]. Units: [seconds].

Note: The total period of the trot gait is $2 * \text{overlap_time} + 2 * \text{swing_time}$ in seconds. So if you wanted a 1Hz gait, you could set `overlap_time=0.25` and `swing_time=0.25`.

1.11 Final Project

Contents

- *Final Project*
 - *Overview*
 - *Previous projects*

1.11.1 Overview

In teams of 3 to 6, you will scope and complete a project of your choice using the pupper robot!

Project requirements

- **Do something your whole team will enjoy!** – keep in mind that team members will be coming from different backgrounds and you won't want to leave anyone out!
- **Focus your project on the real robot as much as possible** – you will have a live demonstration at the end of the course
- **Make your project reproducible** – make sure your work will be reproducible for future teams

Tips

- Get some code running on the robot within the **first two weeks** of your project. You will find the real robot is more interesting and **much more complicated** than the simulated robot.
- Please expect everything to take 3 times the as long as you think. For example, your team will likely not be able to implement MPC or RMA from scratch in the time allotted.

Deliverables

- Project proposal
 - 1 - 3 slides illustrating your intended work
 - We will help you scope appropriately
- Midway update / request for help
 - Progress update for the teaching team so we can help you as best as possible
- Final presentation
 - 5 - 10 minute oral presentation about your project
 - * Project motivation
 - * Approach
 - * Results
 - * Lessons learned / reflections (!)
 - * Class feedback (!)
 - Live robot demonstration
 - 5 mins Q & A
- Report + Github repository
 - In your github README include a 500 - 1500 word version of your presentation that covers the same points
 - Also include instructions that are detailed enough for other groups to reproduce your results

1.11.2 Previous projects

SATE: Simulator Adaptation through Task-Agnostic Exploration

Project report: <https://drive.google.com/file/d/1OYgZvrpDRrE17dB6kOqpYvIy6UJp60Ic/view?usp=sharing>

Pupper Karel API

Project code and report: <https://github.com/stanfordroboticsclub/karel-pupper-api>

Learning Efficient Gait Transitions

Project report and code: <https://github.com/jadenvc/pupper-learned-gait-transitions/>

1.12 FAQ

1. Pupper's Raspberry Pi is not turning on / has some power issue

Some of the older bottom PCBs have broken 5V regulators. Other boards have working 5V regulators that stop working when the Raspberry Pi also gets power over USB. In either case, we recommend unplugging the 5V power to the RPi (the black and red 2-wire cable) and instead powering the Pi via USB C from a computer or portable battery.

1.13 Raspberry Pi Reference

1.13.1 SSH and Internet at Stanford

Note: SSH only works when both the RPi and computer are on the "Stanford network" and in the same building.

1. Connect your RPi to your computer over USB C (both sides) or ethernet cable.
2. SSH into the RPi over the local network `ssh pi@raspberrypi.local` ¹
3. Find the MAC address `ifconfig`. The MAC address is the hexadecimal number in the ether line of the wlan0 section. It looks like 6e:19:c0:ce:a5:22 or similar. [TODO: Insert screenshot].
4. Go to iprequest.stanford.edu on your computer
5. On the first page, choose Device Type: Other, Operating System: Linux, Hardware address: the MAC address you found
6. On the second page, choose Other PC, delete what's under Hardware Addresses Wireless and replace with Pi's MAC address.

7. Wait for email that says the device has been accepted.
8. Reboot the Pi `sudo reboot` 0
9. SSH over cable
10. Connect to the “Stanford” network on the Pi by doing `sudo raspi-config` and following options for wifi.
11. Test the connection by doing `ping google.com` and checking that bytes are received.
12. SSH into the Pi from your computer via `ssh pi@[rpi's IP address]` such as `ssh pi@192.168.9.69`.

Notes:

¹ If you cannot SSH over cable, plug a monitor, keyboard, and mouse into the RPi. Then use `sudo raspi-config` to enable SSH. Then use `ifconfig` to find the ethernet IP address.

1.14 CAD Models

1.15 Assemblies

1.15.1 Pupper

1.15.2 Arm Pair (Static Base)

1.16 Leg Components

1.16.1 Left/Right Lower Leg

1.16.2 Upper Leg

1.16.3 Left/Right Hip

1.17 Body Components

1.17.1 Motor Bulkhead

1.17.2 Electronics Bulkhead

1.17.3 Bottom PCB Cover A

1.17.4 Bottom PCB Cover B

1.17.5 Charging Connector Holder

1.17.6 Leg Pair Assembly

1.17.7 Front Cover (Pi Camera)

1.17.8 Front Cover (OAK-D Lite)

1.17.9 Left/Right Side Cover

1.18 Libraries

BasicLinearAlgebra: <https://github.com/tomstewart89/BasicLinearAlgebra>

DJIC610Controller: <https://github.com/stanfordroboticsclub/DJIC610Controller>

1.19 Actuator troubleshooting

1.19.1 DJI C610 motor controller

[-> Manual <-](#)

Motor does not move

- If the motor controller is blinking abnormally or beeping, read page 6 of the manual linked above
- Power cycle the ENTIRE system (including Teensy and motors)
- Turn the motor by hand to see if you can feel if any of the teeth inside the actuator broke
- Check that the cables from the motor to motor controller are not loose or broken

How to calibrate the motor

1. Arrange the motor so that there is no load on the motor. This means nothing can be preventing the motor from moving, and gravity should not be applying torque to any link attached to the motor.
2. Hold the button on the motor controller down until the motor starts moving
3. Let the motor do its calibration routine
4. Controller should restart and do it's normal green blinking

1.19.2 DJI M2006 motor

[-> Manual <-](#)