Massachusetts Institute of Technology

Robotics: Science and Systems I

Lab 3: Robot Chassis and Simple Commanded Motions

Distributed: Tuesday, 2/17/2009, 3pm
Due: Monday, 2/23/2009, 3pm

Objectives

Your objectives in this lab are to:

- Assemble a functional robot chassis, using your materials from the Motor Control Lab and additional parts we supply.
- Implement odometry, and use it to control simple robot motions and motion paths.
- Implement code to command your robot to translate and rotate by specified amounts.
- Establish coordinate systems for the robot body and the hangar, measure "ground truth" (i.e., the robot's actual physical motions), and evaluate the accuracy of your robot's commanded motions.

This lab connects the concepts introduced in the previous lab, namely motor angular velocity control and encoder sensing, to odometry-controlled robot translation and rotation. Previously, your robot's wheels were commanded either via PWM or at a slightly higher level, angular velocity. In this lab, you must control the robot as a whole, not just its wheels. Because we're dealing with a robot, we wish to have our robot move in terms of *translational* (i.e. *linear*) velocity, and rotate with respect to its body. In this lab, to keep things simple, you will write code to command your robot either to translate (i.e. roll straight forward or backward) a specified distance, or rotate (i.e., turn in place) through a specified angle.

Physical Units

We remind you to use MKS units (meters, kilograms, seconds, radians, watts, etc.) throughout the course and this lab. In particular, this means that *whenever you state a physical quantity, you must state its units*. Also show units in your intermediate calculations.

Materials

To start the lab, you should have:

- Your bench rig from the Motor Control Lab:
  - 2 motors
  - 2 wheels
  - 2 motor screws
  - 1 15" aluminum 80/20 beam
  - 6 1/4-20 x 1/2" hex bolts and sliders
  - 8 8-32 x 3/8" hex bolts
  - 4 8-32 x 1" plastic hex standoff
  - pegboard

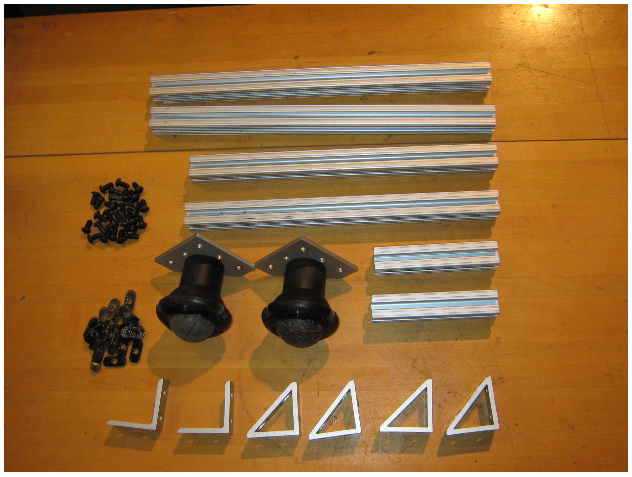

Figure 1: For this lab you should have recieved the materials pictured

  - OrcBoard,OrcPad, AC power adaptor
  - lead-acid battery and power cable
- In addition (see Figure **??**):
  - 2 15" aluminum 80/20 beams (for a total of 3)
  - 2 13" aluminum 80/20 beams
  - 2 5" aluminum 80/20 beams
  - 6 brackets (2 L-brackets, 4 angle brackets)
  - 2 caster wheels on mounting plates
  - 36 1/4-20 x 1/2" hex bolts and sliders (for a total of 42)
- In your bin:
  - Hex keys
  - Tape measure
  - Sharpie marker
  - Masking tape for marking floor grid

Time Accounting and Self-Assessment:

Make a dated entry called "Start of Chassis Lab" on your Wiki's Self-Assessment page. Before doing any of the lab parts below, assign a number to describe your proficiency: 1=Not at all proficient; 2=slightly proficient; 3=reasonably proficient; 4=very proficient; 5=expert

- **Java**: How proficient are you at programming in Java (as of the start of this Lab)?
- **Motion**: How proficient are you at crafting robot motion algorithms using odometry?

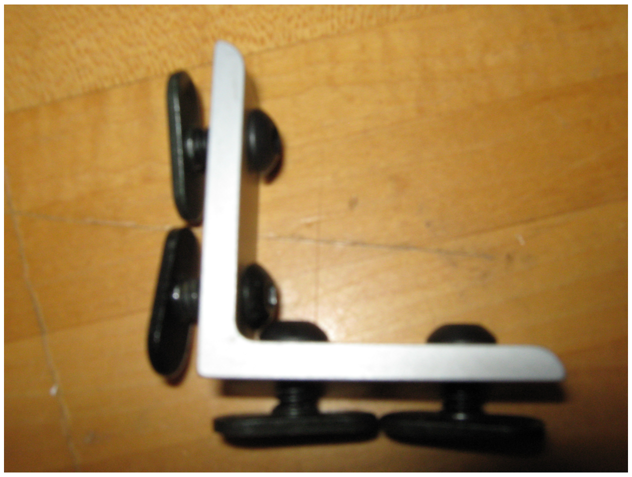

Figure 2: Rollbar assembly: Put bolts and sliders on the L-brackets.

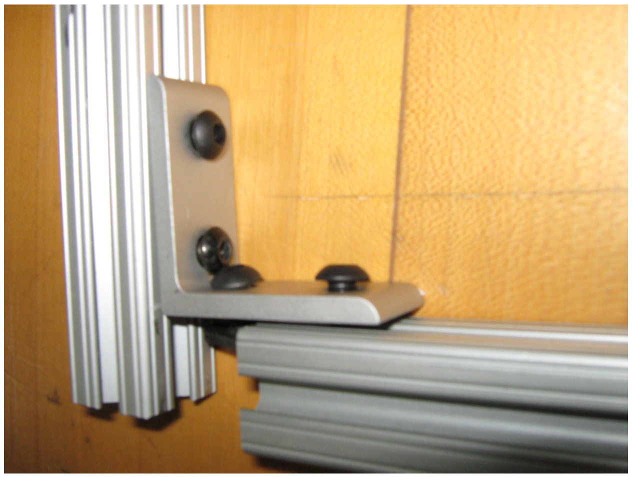

Figure 3: Slide the 5” 80/20 onto the L-brackets.

1 Assemble the Chassis

You will now assemble the robot chassis. You may copy the staff’s exemplar robot, or follow the instructions given below. Some steps, such as 1, 2, and 3, can be carried out in parallel. In general, do not tighten down the bolts until told to do so; this will make your life easier when sliding bars together and adjusting them.

1. Assemble the rollbar.
   (a) Put four 1/4-20 bolts and sliders on each of the L-brackets, as shown in Figure **??**. Do not tighten down the sliders yet.
   (b) Slide the L brackets onto a 15” 80/20 beam. Then slide the two 5” 80/20 beams onto the L-brackets, as shown in Figure **??**.
   (c) Your completed rollbar should look like the one in Figure **??**.
2. Put 1/4-20 bolts and sliders on all the angle brackets, as shown in Figure **??**. Do not tighten them down.
3. Put eight 1/4-20 bolts in the pegboard - one in each of the front corners, one behind each of the two side notches, and two catty-corner on each of the back corners. Place sliders on the ends of the bolts, but again do not tighten them down.

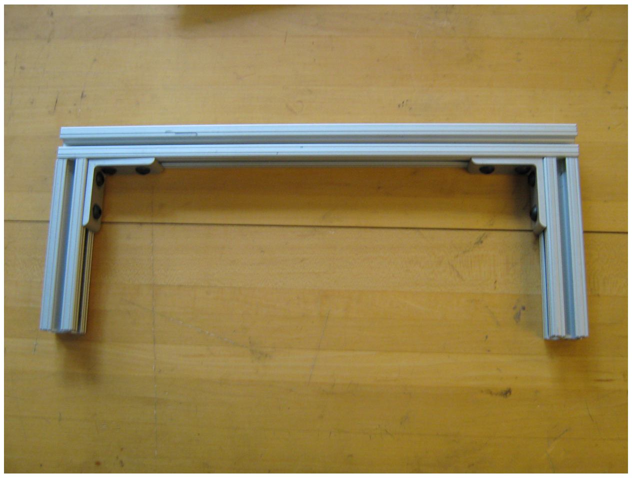

Figure 4: Completed rollbar.

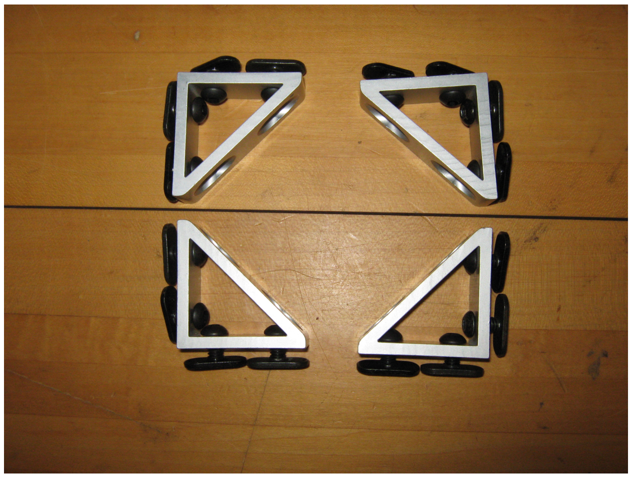

Figure 5: Angle brackets with sliders.

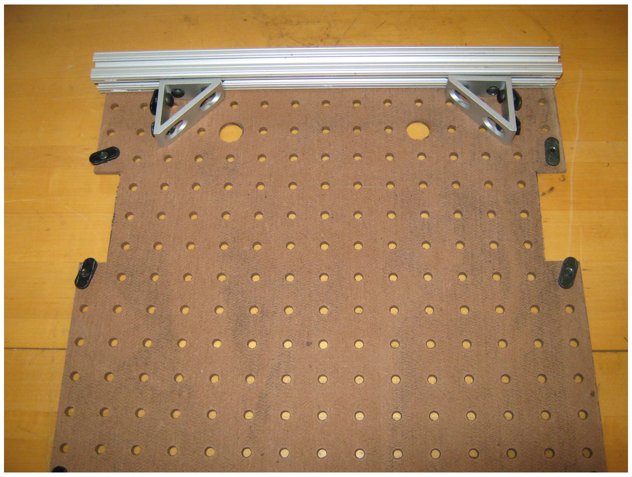
Figure 6: Slide 15” 80/20 and angle brackets onto front sliders. The placement of your sliders on the pegboard may look slightly different from that shown here.

4. Slide the 15” beam of 80/20 onto the two front sliders, as shown in Figure **??**. Slide two of the angle brackets onto the 80/20, placed so that their hypotenuses are facing inwards.
5. Slide the two 13” beams of 80/20 onto the two sides of the pegboard. As you reach the notches on the sides, place an angle bracket there and slide the 80/20 onto it as well. Be sure to orient the angle bracket so that its hypotenuse is facing the back of the robot, towards you, as shown in Figure **??**. As you reach the angle brackets attached to the 15” beam, slide the side beams onto it as well.
6. Place one slider in the top groove of each side beam. Place two more in each of the outside side grooves, for a total of three sliders per beam.
7. Slide the last 15” 80/20 bar onto the two back sliders, as in Figure **??**. Put two sliders in the top groove.
8. Place the casters in the back corners, as in Figure **??**. Attach with 3 1/4-20 bolts each, and tighten them down.
9. You can now tighten all of the 1/4-20 bolts on the chassis, except for those on the angle brackets intended for the roll bar.
10. Flip over the chassis, and attach the rollbar to the angle braces. It should look like that shown in Figure **??**. Move the rollbar and braces as far forward as possible in the pegboard slots, and tighten all the remaining bolts.
11. Mount a motor to each side, using the two sliders placed there previously. Mount them directly underneath the rollbar, as in Figure **??**.
12. Attach the wheels to the motors.
13. Mount your OrcBoard, OrcPad, and battery to the robot chassis. You may mount them wherever you like, but consider the effect of the battery’s weight on the robot’s driving performance. Leave room for the laptop behind the rollbar.

Measure and record your robot’s wheel circumferences and wheelbase (distance between wheels), using the middle of each wheel tread as reference. You should try to ensure that your robot’s wheels are symmetric, i.e. that they are each approximately the same distance from the robot’s center. Also, your robot’s wheels should both be approximately the same diameter, so you may use a single number (both in your report and in your code) for this.

Deliverables: Create a new page on the Wiki, named “Chassis Lab Report Group N” where N is your group number. Link to it from your group’s main page. This report should begin with a brief discussion of your assembly process, especially steps that went particularly smoothly, were particularly difficult, or required substantial rework. Include your values for the wheel diameter, wheel circumference, and wheelbase, as well as a picture of your completed chassis.

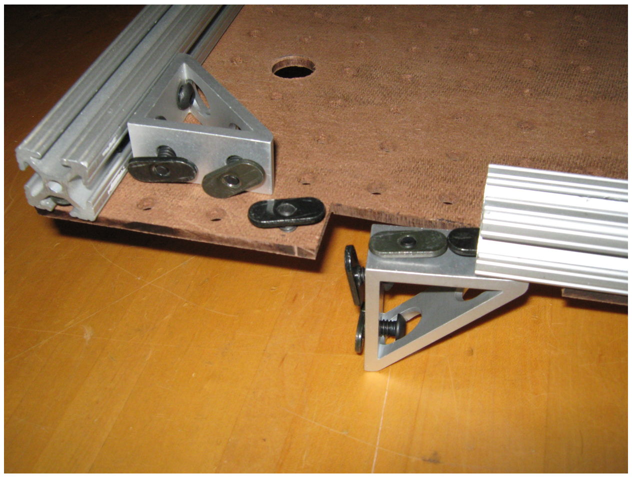

Figure 7: Slide the 13” 80/20 onto the sides of the pegboard, along with the angle brackets that will later attach to the rollbar.

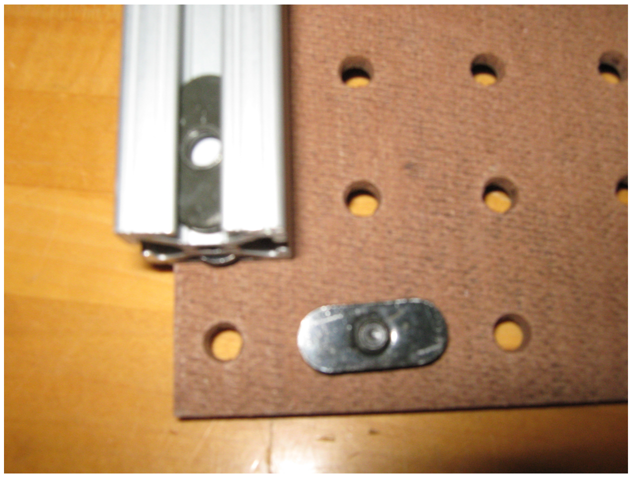

Figure 8: Slide the final piece of 80/20 onto the rear end of the robot.

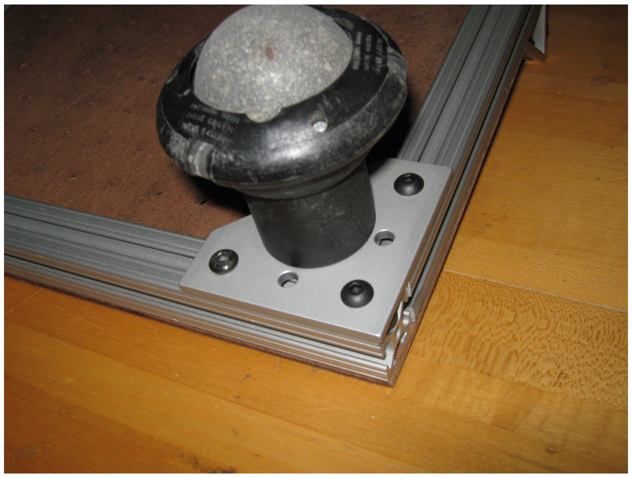

Figure 9: Tighten the casters onto the two rear corners.

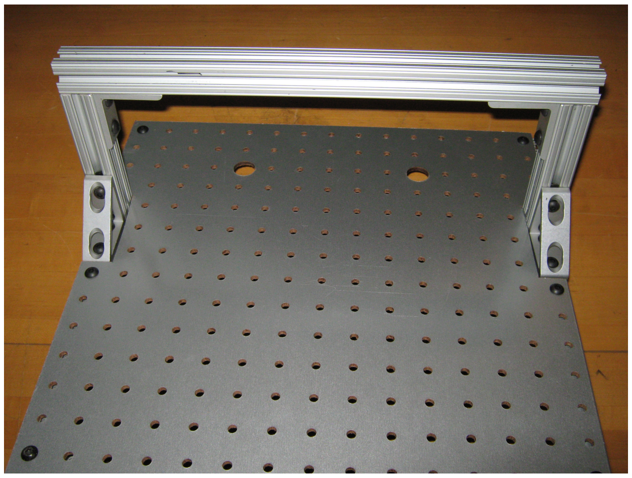

Figure 10: Attach the rollbar to the angle brackets on the top of the robot.

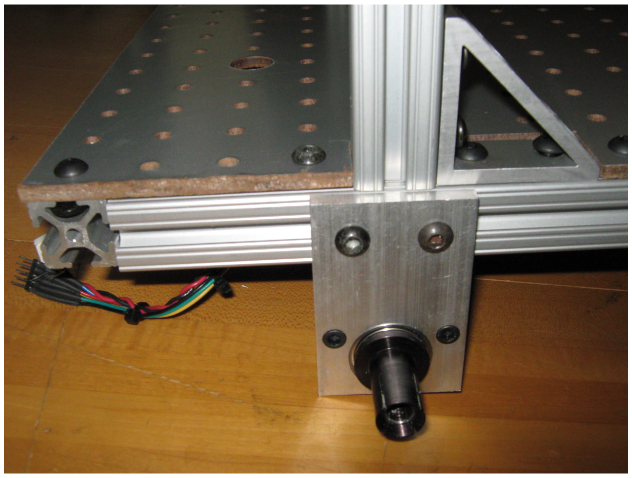

Figure 11: Attach the motors to the side of the robot, directly underneath the rollbar.

2 Get the Chassis Lab Code

This lab requires a working solution for the previous lab: a whole-robot velocity controller which accepts angular velocity commands for each wheel. You may choose either to use your group's solution or the solution we have published.

We have placed solution code to the previous lab (Motor Control) and template code files for this lab (Chassis) in the RSS-I-pub repository. The solution code is available in RSS-I-pub/solutions/. To get the template files, `cd` into your RSS-I-group directory. Type

```
    cp -r ../RSS-I-pub/labs/Chassis .
    svn add Chassis
    svn commit
```

to commit them to your group subversion repository.

Your code for this lab will be written in two files: `Chassis.java`, the lab entry point and motion path specification; and `RobotPositionController.java`, a position controller for your robot. As usual, we have provided base implementations of these files, as well as any other code files you will need for the lab. It should be possible to complete the lab by writing only the indicated code blocks, but we do not require you to stick to this constraint.

3 Connect your Robot to your Workstation

Connect the Sun to your ORCboard with a long serial cable. In this lab, the robot will run "tethered," i.e., with the serial cable connected even while the robot is in motion. One team member will have to act as the *wrangler* to ensure that the robot never outruns its serial cable and encounters damage from the resulting yank. Soon you will integrate a laptop into your robot, enabling it to operate untethered.

Safety Tips

- While debugging your motion code, keep your robot up on the provided blocks, i.e., keep its drive wheels out of contact with the bench. *Make sure that the robot can't jump or fall off of its blocks* while the motors are running.
- Keep long hair, shirt-sleeves, ties and jewelry – and anything else that *could get caught and wound up on a motor shaft* – away from your motors at all times.
- *Enforce a maximum linear velocity, per wheel* of 0.4 meters/second (less than half normal walking speed) in your code.
- When you do deploy your robot to the hangar floor, *wrap the serial cable around its rollbar* a few times so that a cable yank is less likely to damage the ORCboard.
- Finally, *use the red "E-stop" (emergency stop) button* if your robot is about to endanger someone, knock something over, or crash into anything.

4 Verify your Velocity Controller

Make sure that the whole-robot velocity controller you are using, whether it's your solution to the MotorControl lab or ours, is working as expected. For this, re-run the GUI for the MotorControl lab as you did in the last part of that lab. If you have chosen to use our posted solution to the MotorControl lab instead of your own, then you will need to change all the references in your `Chassis` java files from

```
import MotorControl.*;
```

to

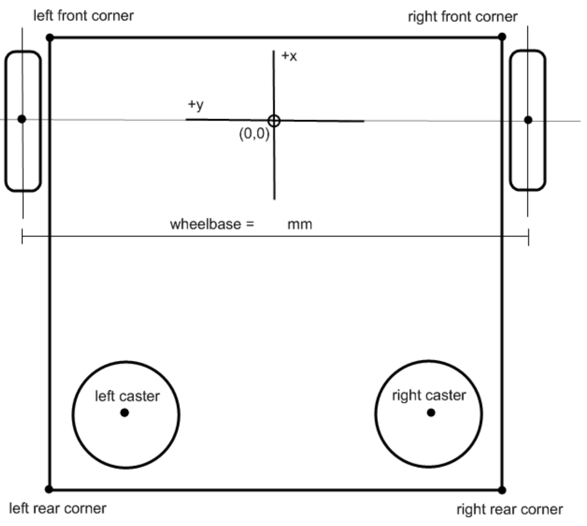

Figure 12: Robot body coordinates.

```
import MotorControlSolution.*;
```

You will also have to add `~/RSS-I-pub/solutions/solutions.jar` to your class path in eclipse.

Run the MotorControl GUI with your robot powered-on, and verify that the robot moves as you expect when you command wheel angular velocities. Do this first with the robot on blocks, and when you are satisfied that is working correctly, with the robot moving on the ground. Test situations with both zero, positive, and negative differential and average angular velocities. Keep the velocities very low at first (about 0.1 rad/s). Adjust gains to try to maximize both performance and stability.

Deliverables: Note your final gain values on the lab Wiki page.

5 Establish Body and World Coordinates

Now establish a 2D "body coordinate frame" for your robot. This frame should have its origin $O = (0, 0)$ at the ground point directly below the midpoint of the robot's drive wheel axis. Your frame should be right-handed, with unit vectors $\hat{x}$ pointing toward the front of the robot and $\hat{y}$ pointing to the left as the robot is viewed from behind. Carefully mark O and (the directions of) $\hat{x}$ and $\hat{y}$ on your robot chassis with masking tape and a sharpie.

Measure and record the positions, in body coordinates, of the two drive-wheel ground contact points, the two caster wheel contact points, and the four corners of the robot's chassis. Make a sketch the robot frame, wheels, motors and casters with each of these points $(x, y)_b$ labeled in the body frame. You will use this information later in the lab to recover the robot's actual displacement and orientation, or "pose" $(x, y, \theta)_w$ in world coordinates.

The TAs/LAs have already marked at least one two-meter square on the floor, in masking tape. This will be the robot's "world coordinates". You may use this square if it is not already taken, with a maximum of three teams per square. If there are not enough squares, you must make your own with masking tape and a tape measure. Take care to make your right angles true, by checking that the square's diagonals have equal length. Subdivide the square into four one-meter squares.

Deliverables: Upload a diagram of the robot to the Wiki, and link to it from your Chassis Lab page. The diagram should include positions of all the components mentioned above, in body coordinates.

6 Translate a Specified Distance

Fill in the `translate` method in `RobotPositionController.java` to command your robot to move forward or backward at a constant velocity for a specified distance in meters, relative to its current pose. *Document your code with comments as you work.* Remember that this velocity is a *linear* measure with units in `m/s` (versus rotational velocity with units of `radians/s`). You may choose to implement either *open-loop* or *closed-loop* control. In any case, your implementation of `translate` should *block*, i.e. it should not return until the translation is considered complete or an un-recoverable error has occurred. The robot should not be moving after `translate` returns.

1. The velocity controller developed in the previous lab expects to see angular velocity of the wheel. Define a constant that converts linear translational velocity of the wheel on the ground to the equivalent (wheel) angular velocity. It should use the wheel circumference, which you measured above.
2. The (signed) number of encoder ticks which have occurred since your program started running will be available in instance variables (read the code to find them). Define another constant that converts ticks to linear translation in meters.
3. If you want to implement closed-loop control then you will probably want to write code both in the `translate` method and in the `controlStep` method. It should be possible to implement open-loop control entirely within the `translate` method. Note that `controlStep` and `update` are both called from the control loop thread for your robot (defined in `RobotBase.java`), and that this is a different thread than the one which calls `translate`. Since states, like encoder tick variables, can be accessed from both threads, you will likely need to use the `synchronized` keyword to reliably read and write states. The synchronized keyword locks a method, only allowing one thread at a time to access it.
4. It is up to you to define what it means for the motion to have "completed" or "errored". One way to do this is to define an error tolerance (in the appropriate units) for the robot position, and to periodically check that the robot is within that tolerance of where you expect it to be. When you expect it to be in the goal position (i.e. the position it should have at the end of the translation) and you check and verify that it is indeed within the defined tolerance of that position, you can consider the motion complete.
5. Wheel slip (e.g. skidding) introduces errors in odometry. If the wheels slip, the robot's motions will be less accurate, and pose error will accumulate. To prevent wheel slip, you need to limit the robot's acceleration. One way to do this is to drive very slowly; another is to explicitly ramp up the commanded speed, over a specified distance, at the start of the motion, and then ramp down the commanded speed at the end of a motion.
6. Having the right parameter values (e.g. control gains) makes the difference between a solution that works great and solution that doesn't work at all. Finding the right parameter values can be hard work. You can make this job much easier by building a user interface for adjusting parameters and viewing the internal state of your solution. You can see how to do this by examining the source code for the MotorControlGUI and OrcSpy applications.
7. Repeatedly place your robot so that its initial pose $(x, y, \theta)_w$ is $(0, 0, 0)$. *Measure the robot's actual final pose (translation and orientation) in world coordinates for a range of at least three combinations of commanded speeds and distances* (add calls to `translate` at the indicated place in `Chassis.java` to issue these commands; you may also want to adjust the controller gains, which are set by code in that file).
8. To measure the actual pose, consider the body frame you established earlier, and work out how a triplet of numbers $(x, y, \theta)_w$ describes the robot's pose (position and orientation) in world coordinates. Given two measured points $(x_L, y_L)_w$ and $(x_R, y_R)_w$ describing the robot's left-wheel and right-wheel ground contacts, respectively, write expressions for the three components of the robot's absolute pose in world coordinates, i.e.

$$\begin{aligned} x_w &= x(x_L, y_L, x_R, y_R) \\ y_w &= y(x_L, y_L, x_R, y_R) \\ \theta_w &= \theta(x_L, y_L, x_R, y_R) \end{aligned}$$

   The wheelbase value(s) measured earlier should match the hypoteneuse of your O - left wheel - right wheel triangle.

9. Repeat each set of experiments at least three times (or as instructed by lab staff). Provide data plots that show the robot's behavior (i.e. plots of error vs. trial for each variable x, y, θ). *Try to explain any significant errors.*

Hint Log your hand-measured (x_L, y_L, x_R, y_R) values to an ascii file. You may use whatever software you like to produce three error vs. trial number plots. If using gnuplot,write gnuplot expressions to compute x_w, y_w, and θ_w from each data entry. The differential error gnuplot file from the Motor Control Lab shows how to express algebraic expressions in gnuplot; the terms \$5 and \$4 in the "`plot`" command refer to the fifth and fourth column of `data.txt`, respectively. Add the line `set angles radians` somewhere above your `plot` command, and use the built-in gnuplot function `atan2` (google gnuplot atan2 for more info). Note that the backslashes in the `plot` command are required when splitting plot descriptors across multiple lines.

Deliverables: Include your plots and explanations in your report, with a brief discussion of the procedure you followed. Document and commit your java code and any gnuplot scripts you wrote or used. Be prepared to demonstrate your robot's motion on the due date marked at the top of this document.

7 Rotate a Specified Angle in Place

1. Fill in the `rotate` method in `RobotPositionController.java` to to rotate your robot about its origin at a specified speed, relative to its current pose. Think about how to encode counter-clockwise and clockwise rotation commands. As with `translate`, `rotate` should block until the motion has completed or errored, and the robot should not be moving after it returns. Again, you may implement either open-loop or closed-loop control.

   Hint Convert the desired robot rotation speed to a linear or angular speed for each wheel, and the desired robot rotation angle to a linear or angular distance for each wheel.

2. By analogy to Part **??**, measure the robot's actual final pose in world coordinates for a range of at least three combinations of commanded speeds and orientations. *Record any undesired translations and/or rotations and try to explain them.*
3. Generate data plots, as above, that show the robot's behavior.

Deliverables: Include your plots, error explanations, and a brief discussion of your procedure in your report. Document and commit your java code and any gnuplot scripts you wrote or used. Be prepared to demonstrate your robot's motion on the due date marked at the top of this document.

8 Driving Out and Back: Error Upon Return

1. Write code in `Chassis.java` to command your robot to drive "out and back:" roll straight one meter ahead, rotate π radians, roll straight one meter ahead, and rotate $-\pi$ radians. In other words, the robot should take a two-meter excursion, returning to its starting pose. You can choose the speed at which, or time duration over which, your robot should execute this excursion. (Test a range of values.)
2. Before running your method, use your results from Part **??** and Part **??** above to predict what your sensed and measured final poses will be after execution. *Write down your predictions in the wiki.* Now execute your program and measure the resulting pose. *How close are your predictions to reality?*
3. Measure the robot's actual final pose in world coordinates for at least three out-and-back trials. *Record any undesired translations and/or rotations and try to explain them.*
4. Generate data plots, as above, that show the robot's behavior.

Deliverables: Include your answers to questions above, error plots and explanations, and a brief discussion of your procedure in your lab report. Be prepared to demonstrate your work on the due date marked at the top of this document.

9 Driving in a Square: Error Upon Return

1. Write code in `Chassis.java` to command your robot to drive twice around a square with one-meter side-length.
2. *Predict and measure the robot's final pose as above.*
3. Repeat this experiment at least three times (starting from the same initial pose each time) and record and plot your results. *How reliably does your robot return to its starting position and orientation?*
4. A camera should be available; if so, capture a short video of your robot's excursion (max size 30MB or as directed by lab staff).

Deliverables: Discuss your robot's performance. Include your data plots and video. Be prepared to demonstrate your work on the due date marked at the top of this document.

Deliverables

This concludes the Chassis Lab. The completed code and Wiki report (with images, plots, and linked videos) are due on the date/time marked at the top of this document. To submit your software for grading, commit all files into your group's repository under a top-level directory named "Chassis". Make sure that a fresh checkout, which the staff will perform, reproduces your code in its entirety.

Time Accounting and Self-Assesment

After preparing your report, return to the Time and Assessment pages of your Wiki. Tally your total individual effort there, including time spent writing up your report. Answer the "Self Assessment" questions again, post-Lab. Add the date and time of your post-lab responses.

Presentation

You may put whatever you feel is relevant in your presentation; these are merely suggestions.

- A brief description of what you did, as individuals and as a team
- Demo video of your robot driving in a square
- Plots from all the experiments
- Any issues and how you dealt with them, problem areas for improvement next year
- Time spent

Wiki Report

Submit the following information on the Wiki.

- Include a discussion of your procedure
- Plot error versus trial number for three different combinations of commanded speeds and distances. Run three trials for each combination, totaling 9 runs.
- Plot error versus trial number for three different combinations of commanded speeds and rotation angles. Run three trials for each combination.

- Plot error versus trial number for three runs, driving one meter, turning 180 degrees, and driving another meter back to the start point.
- Plot error versus trial number for three runs, driving twice around a one-meter square.
- Discuss the plots and error, and provide possible explanations for those errors.