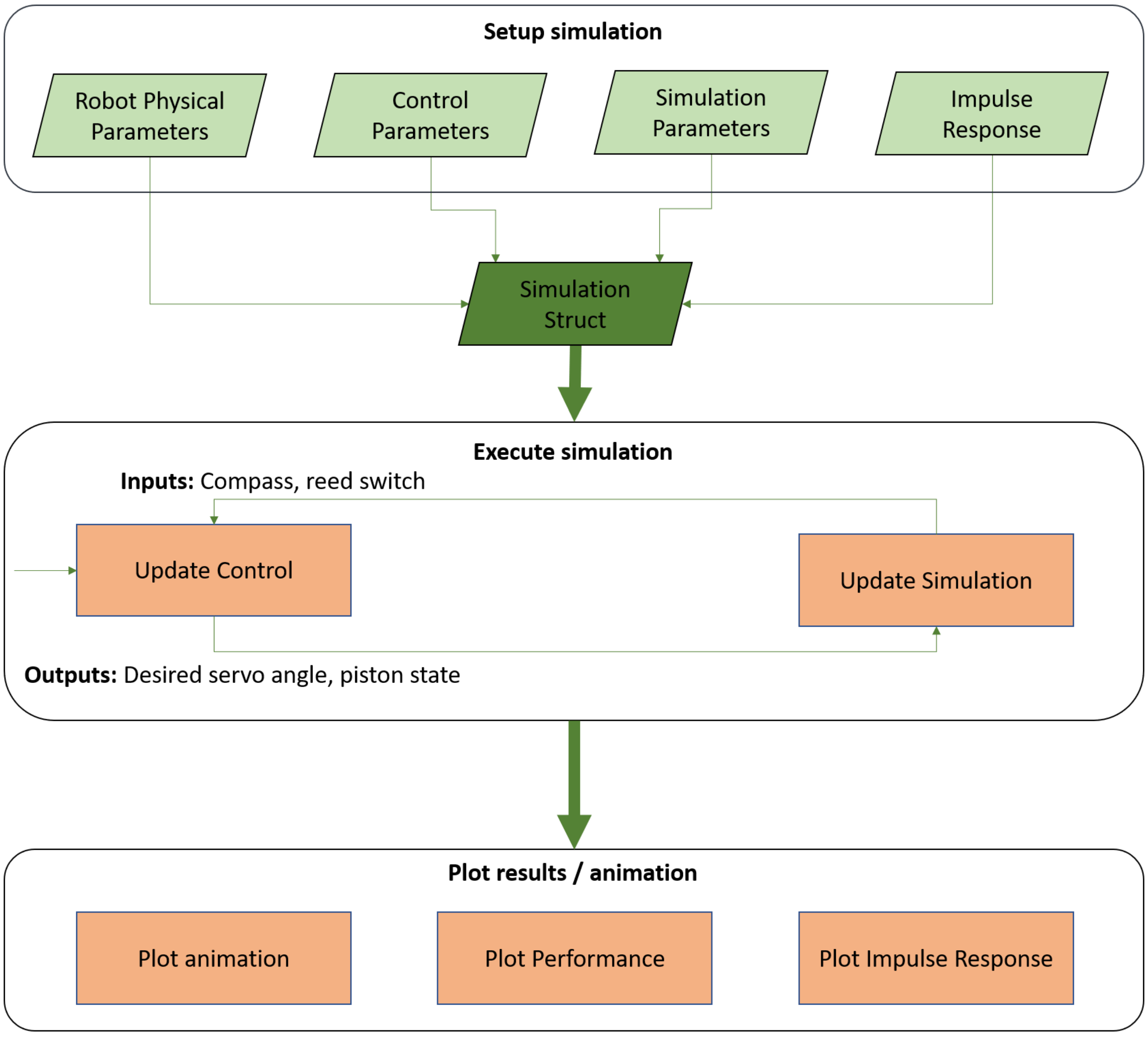
MAE 106 ROBOT SIMULATION GUIDE

Introduction. The MATLAB function robot_simulation.m may be used to simulate the dynamics of a robot during the "Attack on the Anthill" competition, and compare the performance of various control strategies and control gains. This guide will outline how to simulate your robot, how to interpret the various graphical representations of robot behavior, and provide some guidance about how to implement the simulated estimator and controller in your Arduino sketch. The structure of the simulation can be seen in the figure below.



Getting Started. First, open the function robot_simulation.m in the MATLAB editor and press “Run” (shortcut F5). This will run the simulation and open three figures:

Figure 1:

Animation of results. The actual robot position and wheel angle are drawn, as is an outline of the course. The actual robot position traces out a blue line, with blue dots representing the locations at which the piston fires. The position estimate produced by the estimator traces out a red line, with red ‘x’s representing the locations at which the reed switch is triggered by magnets. If using the via-points control strategy, the current target position is presented with the dark grey circle and the other targets in the list of targets is presented with the light grey circles. On the top left you can see the current score of your robot.

Figure 2:

Top – Robot speed. Shows both the actual speed of your robot throughout the simulation and the estimate speed using the reed switch and magnets on the wheel.

Middle – Steering angle. Shows the actual steering angle of your robot throughout the simulation and the expected angle output from your control strategy.

Bottom – Steering error. Show the error in the steering angle that has been added to your servo to make it more realistic. This error is what comes from servo angle error in the controller or changes in the terrain that may cause changes in the position of your steering mechanism.

Figure 3:

These plots in figure 3 come from your data input that will be discussed later in this guide.

Top – Impulse response of position. The actual robot positions measured at different times during the impulse response experiment, with the estimate position plotted over them.

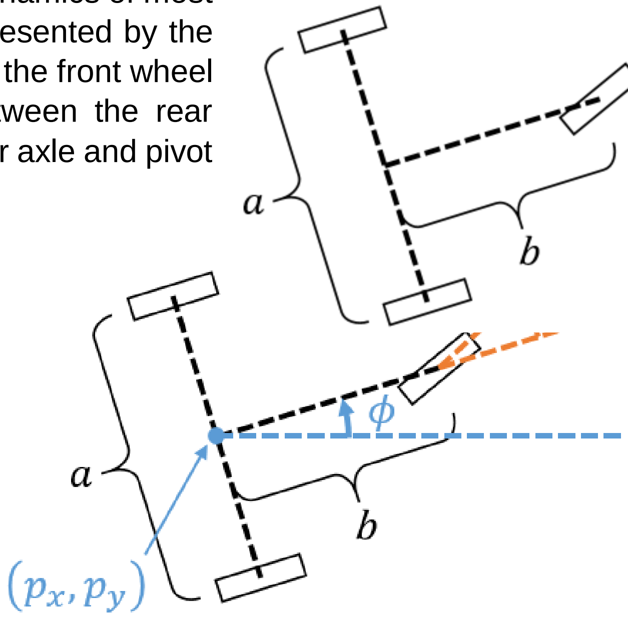
Middle – Impulse response of velocity. The derivative of the estimated position. Each time the piston fires, this pattern is added to the time history of velocity to simulate the superposition of several piston fires.

Exploring different controllers. There are three different controllers implemented in the default simulation:

- Open-loop control: uses number of ticks to decide when to start turning, and time to decide when to stop turning. No sensors used.
- Hybrid control: uses ticks to decide when to start turning, and time to decide when to stop turning. After that, it uses the magnetometer to try to keep the correct direction
- Closed-loop control: uses a via-point strategy with dead reckoning as well as the magnetometer to control the robot.

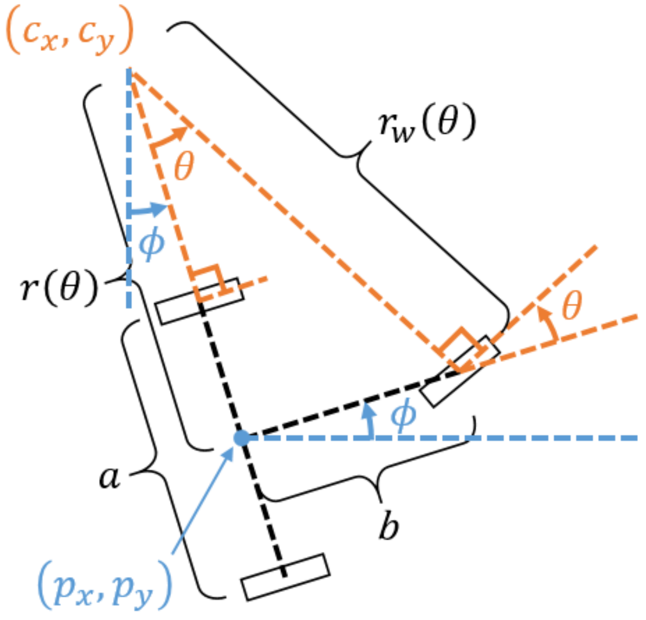
To select the controller you want to use you should change the variable *control_type* in the file "\Setup Simulation\load_control_parameters.m". The parameters for each of the controllers are also in the same file. This file is also the one you create any extra variable that you want to use in your controller. In a parallel with Arduino, this is the place where you would declare your variables in combination with your *void setup()* function.
The actual control for both the piston and the servo motor are in the "\Execute Simulation\update_control.m", and you can look at this function as the *void loop()* in your Arduino. Finally, the implementation of the controllers can be found in the folder "\Execute Simulation\Control".

Robot Dynamics and Estimation. The dynamics of most robots in this competition can be well represented by the dynamics of a three-wheel design in which the front wheel pivots for steering. Call the distance between the rear wheels a and the distance between the rear axle and pivot point of the front wheel b.



Describe the robot's x-y position on the course $p = (p_x, p_y)$ as the position of the center of the back axle. For the simulation, the x-axis is defined as the one parallel to the channel with the point $(0,0)$ lying in the middle of the opening in the trench wall. Describe the **robot's direction = $\boldsymbol{\phi}$** as the angle between the robot's forward direction and the x-axis. Finally describe the robot's **servo angle = $\boldsymbol{\theta}$** as the angle between the front wheel's orientation and the orientation that would cause the robot to move in a straight line.

Assuming that wheels do not slip, the governing principle behind the robot's kinematics during steering is that all of its wheels are constrained to move in the directions perpendicular to their axles (i.e. no slip). This means that when the robot moves, it will rotate about a point $c = (c_x, c_y)$ such that all wheels move along circles whose centers lie on that point. To determine the radii of these circles it is helpful to observe that they represent sides of a right triangle whose angle at c equals θ. The radii of these circles can therefore be seen to depend on θ according to the simple trigonometric relations:



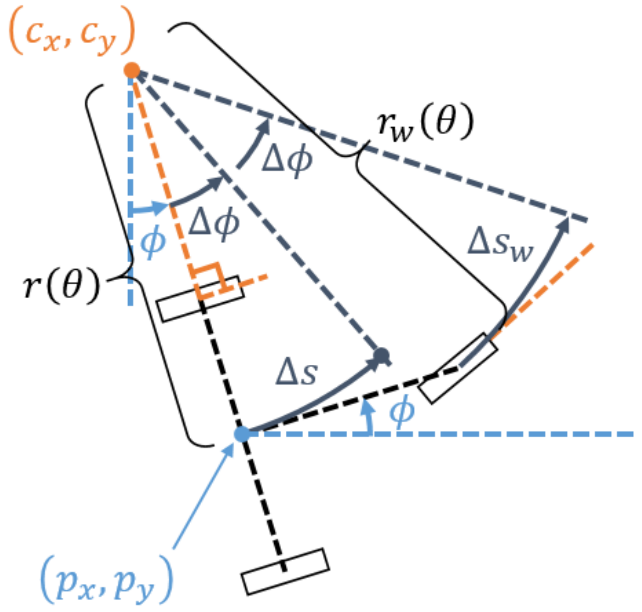
$$r(\theta) = \frac{b}{\tan(\theta)} \ldots (1a)$$

$$r_w(\theta) = \frac{r(\theta)}{\cos(\theta)} \ldots (1b)$$

It is also helpful to observe that the angle at c between the y-axis and the rear axle equals ϕ.

Because the robot is constrained to move in a circle around c, movement of the robot can be simulated from the robot's steering angle and the rotation of one of the its wheels. The simulation requires update rules for p and ϕ so that with each increment in time Δt, it can determine the robot's new position and orientation based on its previous position and orientation.

We know that as the robot moves, the point p will trace an arc (on a circle with radius $r(\theta)$ centered at c) whose central angle equals the change in the robot's orientation. If we determine the arc length Δs traced in an amount of time Δt, we can derive the update rules for p and ϕ with ease. The simulator uses the experimentally measured impulse response of the robot to predict the speed of the robot, v, in response to firing the piston. Δs can then be found by dividing that speed by the sample time:



$$\Delta s = v\Delta t \ldots (2)$$

Because for a hopper, the reed switch will likely be placed on the front wheel, which never leaves the ground, it can be helpful to relate Δs_w traced by the front wheel to Δs.

The arcs depicting the movement of p and the front wheel have lengths Δs and Δs_w respectively, and have the same central angle $\Delta\phi$ equal to the change in ϕ that has occurred since the last update. Therefore, these arc lengths follow the proportionality relationship:

$$\frac{\Delta s}{\Delta s_w} = \frac{r(\theta)}{r_w(\theta)} \dots (4)$$

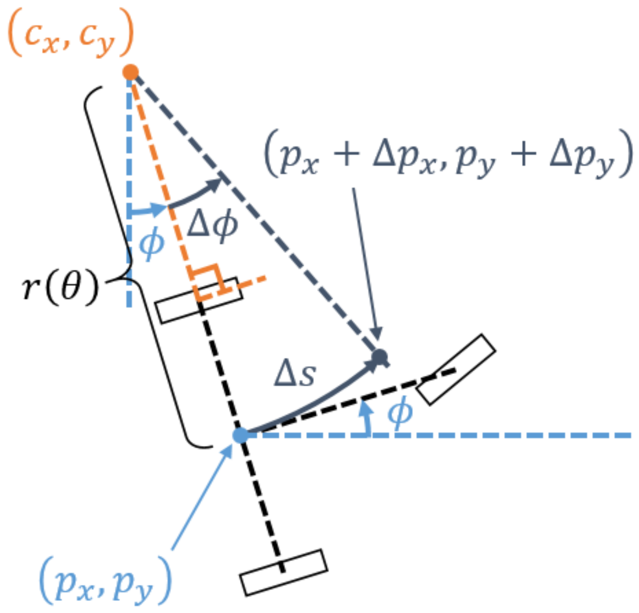
Solving and substitution yields the relationship:

$$\Delta s = \Delta s_w \cos(\theta) \dots (5)$$

The update rule for the robot's orientation can be found using the relation between central angle, arc length and radius:

$$\Delta\phi = \frac{\Delta s}{r(\theta)} \dots (6)$$

Finally, the robot's position p relative to c can be found before and after the update, and subtracted:



$$(p_x, p_y) = (c_x + r(\theta)\sin(\phi), c_y - r(\theta)\cos(\phi)) \dots (7)$$

$$(p_x + \Delta p_x, p_y + \Delta p_y) = (c_x + r(\theta)\sin(\phi + \Delta\phi), c_y - r(\theta)\cos(\phi + \Delta\phi)) \dots (8)$$

Subtracting equation 7 from equation 8 yields the update rule for p:

$$(\Delta p_x, \Delta p_y) = (r(\theta)[\sin(\phi + \Delta\phi) - \sin(\phi)], -r(\theta)[\cos(\phi + \Delta\phi) - \cos(\phi)]) \dots (9)$$

Equations 6 and 9 together can be applied each time step to model of the robot's position and orientation in response to any pattern of piston firing. Although steering angle may change between time steps, making Δt small will generally lead to good approximations. Due to the singularity in equation 1a at $\theta = 0$, simple update rules for straight line motion must instead be used when the robot is traveling in a straight line:

$$\Delta\phi = 0 \dots (10)$$

$$(\Delta p_x, \Delta p_y) = (\Delta s\cos(\phi), \Delta s\sin(\phi)) \dots (11)$$

For four-wheel designs or designs with the reed switch placed near a rear wheel, equation 5 would need to be replaced with the appropriate relationship. For four-wheel designs θ might refer to the orientation of one of the front wheels or be experimentally related to servo motor angle by measuring the resulting radii of rotation for various servo motor angles.

Estimator. Wheel rotation can be measured by counting how many times a reed switch mounted near a wheel passes magnets attached to the wheel. This measurement has finite resolution, limited by the number of magnets (e.g. with four magnets the resolution is a quarter of a revolution). Note that for a three-wheeled hopper design it is preferable to measure rotation of the front wheel, as the rear wheels will likely hop off of the ground, and their rotation will not well reflect the movement of the robot. Our goal is to develop an update rule for p so that each time a magnet passes the reed switch, we can estimate the robot's new position. The robot's orientation can simply be estimated using the compass.

Equations 5 and 11 create a sufficiently strong estimator of p given rotation of the front wheel and steering angle. In this case, Δs_w and Δs can be regarded as the distance traveled each time a magnet passes the reed switch.

Small errors in these update rules will accumulate over time, particularly given that θ is measured from the angle input to the servo and not from direct measurement of the steering angle. This error can be mitigated by filtering the signal from the compass, to produce a more accurate, less noisy, estimate of orientation, which in turn leads to an improved estimate of the robot's position.

Entering Robot Parameters. There are numerous parameters in the load functions (inside the *Setup Simulation* folder) that you should enter to cause the simulated dynamics to closely match true dynamics of your robot. Parameters labelled [MEAURE] should be quick to measure and enter. Parameters labelled [EXPERIMENT] will probably require some amount of experimentation to tune the simulation to match your robot. Parameters labelled [SELECT] are for you to decide what values to use, such as selecting and tuning a control law. Try different options and see what works best.

Simulation Features. The simulation has a few additional features that will help test the performance of your robot during the competition. First the simulation can randomly place obstacles (stationary opposing robots) that can be detected when within some distance of your robot. By default, the controller turns sharply to the left or right to avoid obstacles on the right or left respectively. The advantage of using feedback control is that after avoiding an obstacle, the controller will guide the robot back on track.

The simulation also models the stochastic nature of the robot dynamics. First, the simulation applies a moderate drifting error to the robot's steering angle. This represents the likelihood that the relationship between the servo motor input and the steering mechanism angle is not perfectly consistent. Second, the simulation applies a random error to the magnetometer angle. This prevents the magnetometer from being useful as a perfect measure of robot orientation, representative of the limitations of the of the actual magnetometer.

Calibration Experiments. One important step in calibrating your robot is to measure its impulse response. In this context, an impulse response can be described as the velocity curve that results from a single firing of the piston. If the robot behaves like a linear system, then adding multiple instances of the impulse response (one each time the piston fires) should approximate the robot's true velocity. This is actually how the simulation models the robot's velocity.

To calibrate the simulation with the impulse response of your robot, write a short Arduino code that fires the piston once and records the time whenever a magnet passes the reed switch until the robot stops. Then enter the times into the MATLAB function "\Setup Simulation\load_impulse_response.m" in the section labelled "Impulse Response (velocity profile created by one pulse)". Enter these times as a row of the matrix *impulse_response.data*. You can enter as many rows as you collect (if the rows are different lengths, add NaN values at the end of the short rows until they are all the same length). The simulation will then combine these results and calculate the average impulse response.

If your robot's impulse responses do not add linearly, you can apply a scaling factor to the impulse response (*scale_impulse_response*) so that the max speed predicted by the simulation matches that of the actual robot. Simply calculate your robot's max speed, simulate the robot moving at max speed and set the scaling factor to the ratio of the two. Most robots also slow down as the air tank pressure drops. To simulate this behavior, the impulse response can be scaled by a function of how many times the piston has fired (*decay_impulse_response*). Currently the simulation assumes a quadratic relationship between number of fires and scaling factor until the factor decays to zero (after 56 fires).

Implementing the Estimator and Controller in your Arduino Sketch. For the most part, the code in the simulation's estimator and controller blocks can be translated into your Arduino program. Some of the math is trickier to program in Arduino than in MATLAB so be patient and look up how to use different mathematical and trigonometric functions. As part of your controller in Arduino, you will need to implement the sequences of targets you use in the Matlab simulation. It will help to make sure that you understand what is happening in MATLAB before you transfer a line to Arduino, and think about ways to us the Serial Monitor to test that your estimator and controller are working as expected. Best of luck, and remember, Google is your friend!