**Rework existing robot control code**

**Motor control - Done**

* Write function that accepts linear and angular velocity and sets each wheel velocity
* Angular velocities of each wheel are linearly mapped to PWM outputs. Mapping does not need to be precise, but close enough to approximate speeds with real units making it easier to debug

**Encoder readings - Done**

* Hall effect sensors output a 1 with a north and 0 with south so the interrupts are set to measure changes in the reading for 192 ticks/Pi rad
* Use encoder reading code to calculate angular velocity at max and min motor driver values to calibrate the angular velocity map
* Use the encoder ticks to calculate the change in x distance, y distance and theta (assuming constant velocity in-between iterations and linear velocity in the direction of theta, the kinematics are easily calculated with known radii of the wheels and distance from centre to the wheel)

**State estimation with sensor readings**

* Use gyro and accelerometer with magnetometer to develop Kalman filter to get a more stable/ accurate position and heading estimate
* Magnetometer signal might not be useable because of the proximity to the magnetic encoders

**Combine Kalman filter estimation with encoder estimation to determine position and heading of the agent**

* Use a weighted average to help correct each estimate/ reduce drift and noise

**Update position function - Done**

* Use encoder and sensor readings to update the position of the robot on each loop iteration

**PID control for determining linear and angular velocity - Done**

* Position error is multiplied by the cos of the heading angle to prevent moving in an unproductive direction
* Error is tracked for position and heading to create two separate PID control systems for each agent

**Write move function - Done**

* Move function combines all other functions to move the robot to a given (x,y) position within a specified error

**Write documentation explaining what all the hardware is and how it is connected to the Arduino as well as how the robot control functions are working**

**Wireless networking**

**Use XBees or find other option to send each agent a target position and then have them send a signal when they have reached the position asking for the next position**

**XBees**

* XBee has an API mode that allows sending messages to individual radios instead of all available. There is a Java, Arduino, and Python library for sending and receiving API mode messages
* Set up XBee radios for communication with each other
* Java library does not compile on 64-bit Windows because of a plugin but likely will on 32 bit or Linux
* Python library works for API mode
* Arduino library is not working for communication with example code (other people seem to have the same issues and the library hasn’t been updated in 6 years so it is likely an issue with the library)

**Wi-Fi module**

* Setup local webserver on the host computer that the Arduinos can connect to and send and request data (positions to go to, localization messages, updates on where each agent is for debugging)
* <https://www.instructables.com/Arduino-Esp8266-Post-Data-to-Website/> link to a similar project
* Setup HTTP GET commands so the host computer can send localized position of each agent and send the next position for the robot to go to
* Setup HTTP POST commands so the Arduino can send its state estimation and other data mostly for debug purposes
* Would require a cheap router for operation and a Wi-Fi module for each agent
* Popular module is the ESP8266 ~$15

**Rewire robots so cable runs are cleaner and more organized**

* With sensors attached and working, re-run the connections with wires cut to size and zip tied to the chassis
* Make bus for ground and 3.3 V connections
* Maybe change the power switch for one of the smaller circular switches recessed into the chassis (current one is big with long cables)
* Remove the screw terminals

**Hard mount components**

* Mount the IMU with screws so noise isn’t introduced by movement of the unit
* Mount the wifi module
* Improve the Arduino mount to be more permanent

**Fix/rework/rethink localization beacons**

**Beacons**

* Come up for standard to encode the time of message in the signals along with a checksum or something similar to ensure the message is correct
* Write Python code to send the IR signals
* Write Python code for sending the US signals
* Setup code for receiving the IR signal (might need an Arduino with additional interrupt pins or the sensor would need to be checked on each iteration)
* Write Arduino code for receiving the US signals
* The Arduino would need to send a signal successfully received message to the host computer if the checksum was correct

**Cameras**

* Cameras could be a good option, but the host computer would need to be more powerful that an RPi
* Cameras would need to be corrected for lens distortion and fiducial markers could be used to mark coordinate positions on the floor and each agent to make it easier to recognise landmarks and calculate positions
* Camera would likely need to be mounted high for operation
* The code would likely be much simpler than the beacons since all the work is being done by the open CV library and does not need to be developed from scratch
* A research project was done using the same webcam I have, and they reported good results
* Cameras would also allow the program to be run of any computer instead of requiring a raspberry pi for GPIO pins

**Operation Loops Pseudo Code**

Arduino Side

{

Get initial state info;

While(not at final position)

{

Get target position;

Drive to Target Position;

Ask for updated localized position;

}

}

Host side

{

Read csv into memory with xy for each agent in time (2 by N by t array);

Use localization to get initial states of each agent;

Send initial states;

While(not at final position)

{

Wait for position request;

Send next positions;

Wait for localization requests;

Locate each agent;

Send accurate position estimates;

}

}