Volume

1

FAMU-FSU COLLEGE OF ENGINEERING

Team 311: Material Handling Robot

Members: Alex Jen, Tony Zhang, Walter Caldwell, Sebastian Muriel

Operations Manual

Table of Contents

Chapter 1

Project Overview and Components List

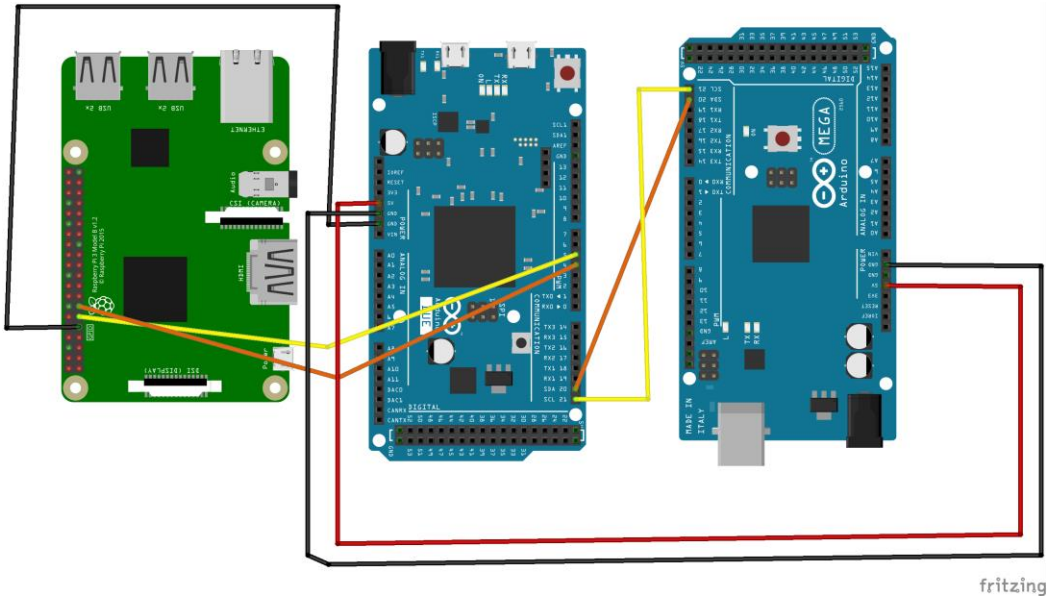
This project consists of two teams an ECE/EE team and a ME team. Our project was to create a partially autonomous robot that operates in the dark of a warehouse. It would use IR black and white lines to sense the ground, a pulley forklift for lifting packages, and keep track of items in the warehouse with its internally stored database. The Mechanical team worked on designing and creating a chassis, lifting system, and getting the PD omni-wheels controls set up. The ECE/EE team worked on developing a system for the robot to receive input from humans, the navigation of the warehouse, using the QR sensors, integrating the pulling, and the PI controls for the line following.

The components used on our side of the project are:

1. Raspberry Pi 4 w/ 64-bit Ubuntu
2. Arduino Due
3. Arduino Mega
4. Raspberry Pi Camera Module V2-8 Megapixel,1080p (RPI-CAM-V2)
5. Arducam Pan Tilt Platform for Raspberry Pi Camera, 2 DOF Bracket Kit with Digital Servos and PTZ Control Board
6. QTR-8RC Reflectance Sensor Array

Our GitHub can be found at: Sebas-Muriel/WarehouseRobot (github.com) . In this GitHub it contains our website, and all the files for starting the NodeJS server and contains all the code for the Mega, Due, and Pi.

Wiring Diagram



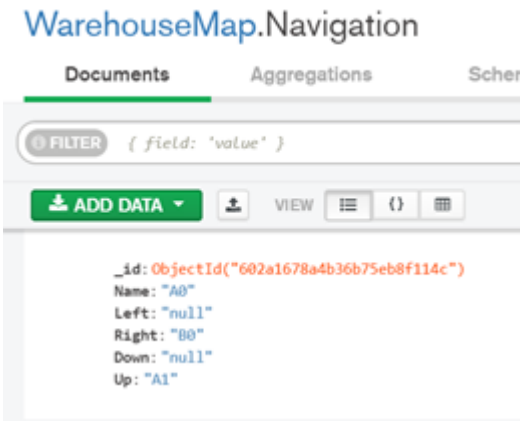
For the main connections the Raspberry Pi's GPIO17 and GPIO24 must be connected to the motor controller for the handshaking. In addition, a ground wire must be connected. The Pi is connected to the Due via the UART usb serial port. The only connections missing are from the Arduino Mega IR sensor pins. These are shown in the code for the pins that must be used for them.

.

MongoDB

For keeping track and storing the items in the warehouse, the Raspberry Pi hosts a MongoDB database server on it. MongoDB is a non-SQL database that stores information in JSON style. MongoDB should start up when the Pi boots up but if not, there are plenty of resources online for the commands to start MongoDB. If you want to install a new copy on a Pi you must follow the instruction from MongoDBs website. For our project we have two collections inside the database: Navigation and QRs. The database name is called WarehouseMap as well. Navigation contains all the intersections of the warehouse. They are structured as the picture below.

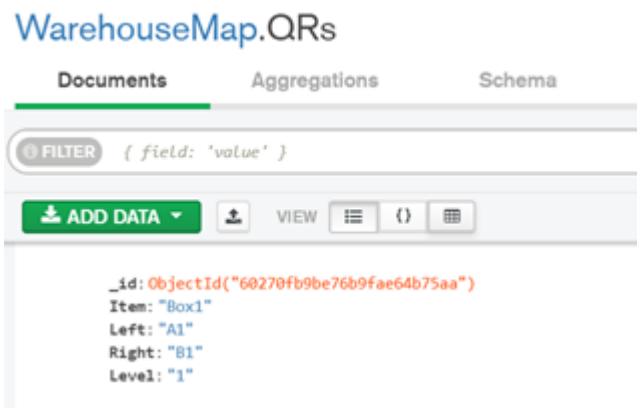
Navigation



It contains the Name of the node given as a column and number set. A corresponds to the column and 0 corresponds to the row. More on the warehouse design can be found on the warehouse map tab. If there is no path in a direction it is given a null string.

QRs

The QR collection is structured as below:



Here we can see the item name for testing purposes it has been called Box1. Inside this JSON is contains the left node and the right node. This is the row that the package is between. It also contains a level, so the robot does not need to look through however many levels to find the package. Level 1 denotes the ground level and each level after is a shelf up.

MongoDB CRUB operations are used in the NodeJS server that is hosted on the Pi. It is also used in the python scripts to add items and delete items from the database.

If you wish to test MongoDB on your own personal computer instructions are given inside the readme file in the GitHub that explains how to start MongoDB once downloaded.

NodeJS Server

The NodeJS code can be found inside the UserInput file on the GitHub. Running the test.js file will create a server the local host port 8081. Inside the readme file there are instructions on how to IP tunnel the server to allow external access using ngrok. Inside the folder there are also html documents that are for the webpages used inside the program. They are copied and pasted in the program and have some elements changed, so they are not the same as the ones inside the program. You will not need to download any additional libraries to run the server if it is downloaded from the GitHub.

Mongoose:

```
//*****************MONDO DB STUFF***************//
const uri = "mongodb://127.0.0.1:27017/WarehouseMap";
const mongoose = require('mongoose');
const { response } = require('express');

mongoose.connect(uri, {useNewUrlParser: true, useUnifiedTopology: true});
const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function() {
  // we're connected!
  console.log("Connected to Database");
});

const { Schema } = mongoose;

var QR = mongoose.model
  ('QR', new Schema({Item: String, Left: String, Right: String}), 'QRs');

// QR.find( function (err,QRs){
//   console.log(QRs);
// })
//******************END MONGO DB TEST*************//
```

Here is where the program connects to the MongoDB server. If the database is changed the uri will also need to be changed. At the end, WarehouseMap must be the database name choses. Also mongoose uses Schemas to load JSON files in. If you wish to receive more information from the JSON document then more attributes will have to be added. Currently only Item, Left, and Right are the attributes read in.

Get Responses

The server hosts different webpages using the express library an example is here:

```
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname + '/userWeb.html'));
})
```

The homepage of the server is denoted by a '/' with nothing after it. This homepage is using a get response to load up the userWeb.html in the same folder as the server. The userWeb contains buttons and different forms that correspond to different '/'s in the program. There are currently only three pages /pickup, /training, and /stocked. These

specific strings are from the userWeb.html and can be changed if needed. However there needs to be a response from the server or else it will be a blank page.

The three different pages fo the following:

Pickup

Given an item name it will call the pickup python script in the folder with that name as the argument value. If the package does not exist in the warehouse it will give an error to the user. If it can pick up the package it will give a validation message to the user.

Training

Given a uniform grid with no holes inside it, training will insert all the nodes into the database and then perform a scan of the warehouse to insert all the items into the database. The form of the input must be in "nxm" n and m being an integer. This will call the training script in the folder.

Stocked

Clicking this button will scan the database and output all the items that exist in the warehouse as well as the number of the items.

IR Sensors

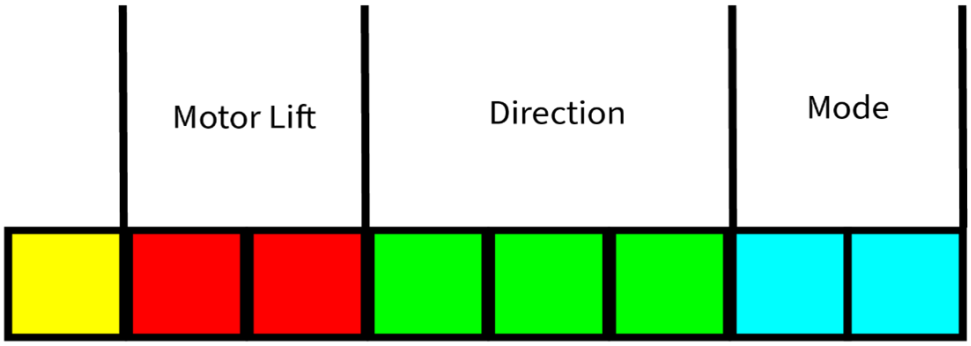
The QTR-8RC sensors are used to read in the black and white colors on the ground. The QTR library is being used to read them inside the I2C_IR Arduino code, which can be found inside the Motor Controller folder. This code contains an address used for the motor controller and on repeat just reads in the sensors and converts them into an 8-bit number and then stores them into a byte array. The IR sensors are all connected to the Arduino Mega which is connected via I2C to the Arduino Due.

Motor Controller

The Arduino Due is being used as the motor controller for the robot. This was picked because of its many interrupt pins for the omni-wheels. The PD omni-wheel code is loaded onto the Due, for more information about that set up look at the T506's operations manual as they designed that code.

Instructions

The Motor controller asked for 4 bytes from the Arduino Mega which responds with the byte array described above. Then the using those values the line sensing PI controller works to center the robot changing the desired robots x,y, and heading angle values. In addition to following the line it also receives instructions on the direction to move in from the Raspberry Pi. The instructions are formatted as below:



There are three modes currently being used. Reset, Node, and Tick mode. The values for these can be found in the defines of the code as well as the defines for the directions and motor lift. The directions used are left, right, up, down, and stop. The motor lift instructions are base level, second level, and lift. Then there is one extra bit to be used for any additional instructions. The instructions being sent to the motor controller are in each python script and it is how the robot moves in the correct direction.

PD Line Sensing

While moving in a certain direction it will only consider two of the IR sensors. The PI controllers are located in the .h file and are called IR_PID. This returns a value that changes the x or y velocity and it also works to correct the heading angle.

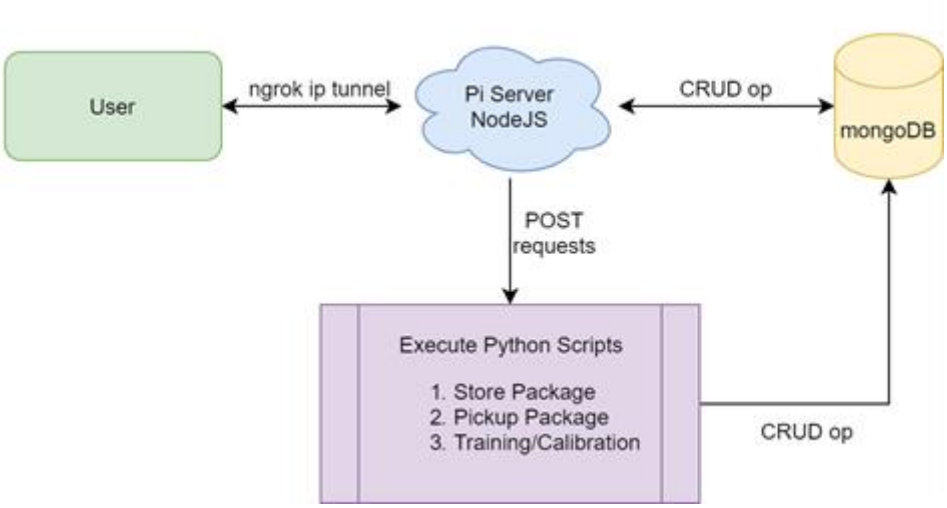
The instructions being received by the Pi are done via pseudo-handshake. Meaning that Pi's instructions take priority of the Arduino's controls. When the Pi wants to send an instruction it beings MRDY low and then the Due responds with bringing SRDY low and then the message is transmitted and parsed. Once the transmission is done MRDY goes high and then SRDY goes high. For some reason this only works successfully on multiple sends, so the code to send in the Pi sends 3 times in .003 seconds.

When the Arduino senses an intersection, it will continue moving until it reaches the center of the robot. Once it has, it will send a "1" to the Pi which will then give it its next instruction. This sending is not done via handshake it sends at will with a new line at the end of it. The Pi continuously reads the UART channel until it receives a 1 for the next instruction.

Python Scripts

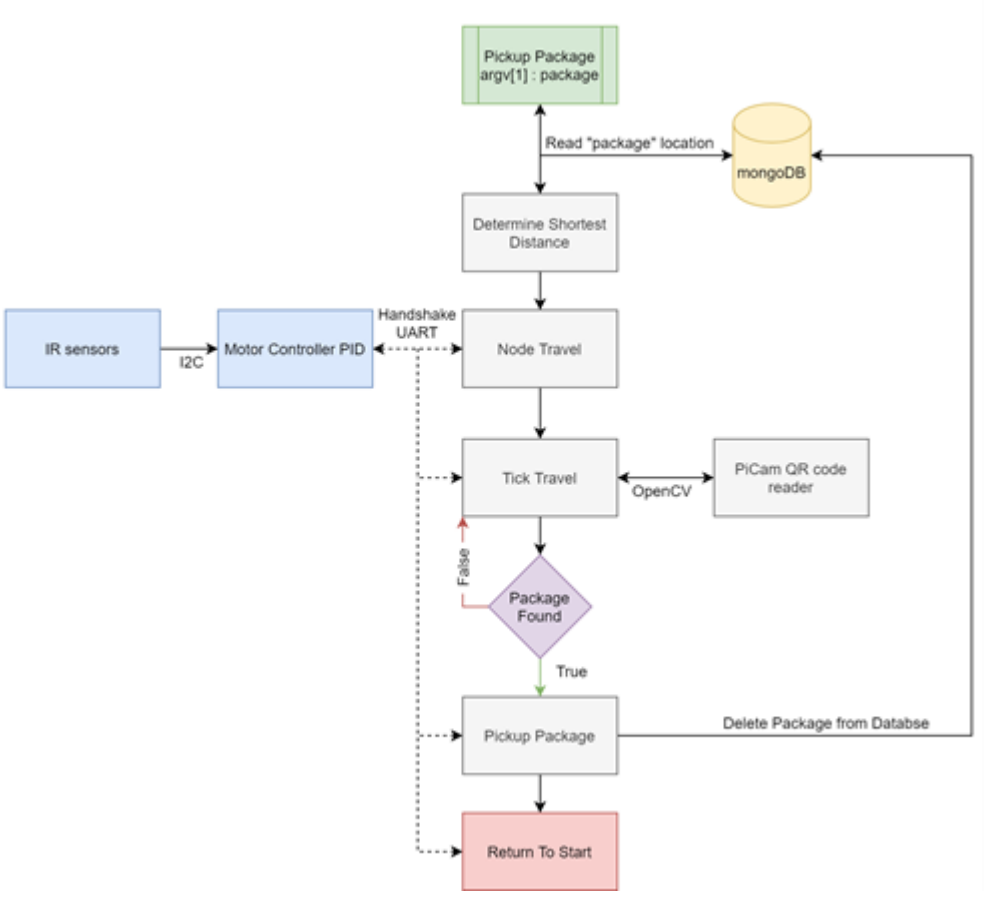
The python scripts send directions/modes/actions to the motor controller for what it is supposed to do. Currently we are only working on two out of the three scripts from the flow chart. We are working on pickup and training. The scripts connects to MongoDB to either add documents to the database or create/update documents to the database. The python scripts also read from the motor controller whenever it encounters a tick or node. A tick is half of the IR sensor being covered while a node is the entire IR sensor being covered. Node travel is used for getting to the correct

horizontal section while tick travel is used for precise movement between nodes. Inside the python scripts the value "A2" is converted into a grid form [1,2]. Then that value is used to calculate the shortest distance if we are picking up a package. The high level flowchart is shown below:



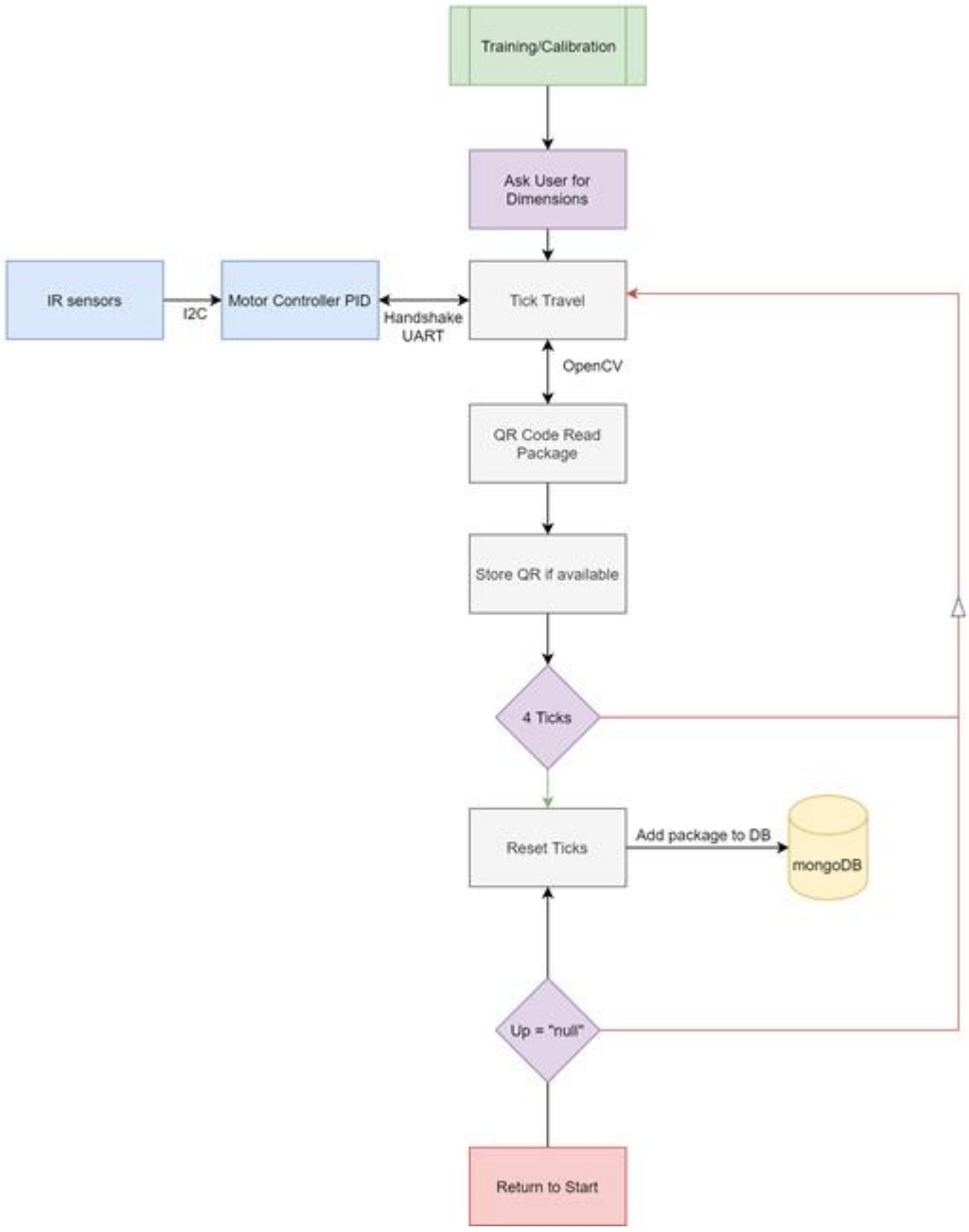
Pickup Package

For picking up a package it takes an argument that tells it what package to pick up. This code is inside the Navigation folder and is called Nav.py. It reads in the item and finds the closest distance from the start position and then goes to the closest node. Once at the node it begins tick movement and searched for the correct box. It also knows the level so it only needs to look at one level. It uses the Pi cam to find the package name. The name of the packages are stored in JSON style: {Name: "Box1"}. There is a 5 second timeout window to read the package this can be changed in the code by changing the integer in the function. Once the package is picked up by sending the motor controller a series of codes, it brings it to a loading bay where the availability is stored in a MongoDB collection (WIP). Once delivered it will return back to its start position.



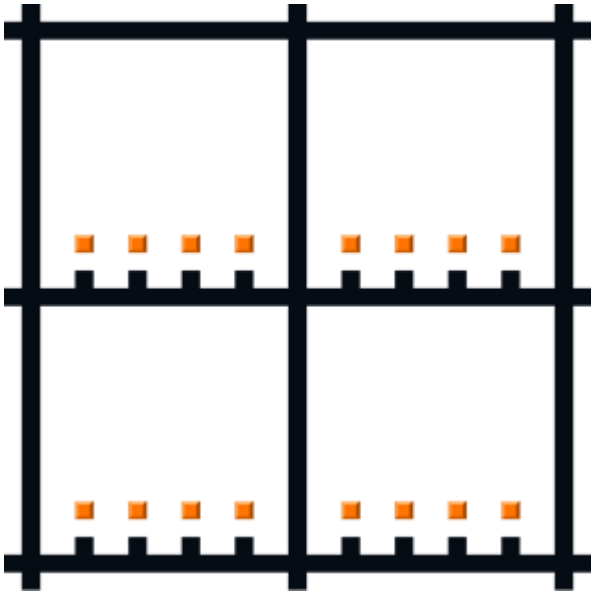
Training

The training script will first convert the grid into nodes and insert them into the database. It will also destroy the current items and grids inside the database. Then it will go through the database left to right reading in the packages and storing them inside the database. Once it's finished it will return back to start to await further instructions. The logic is seen in the flowchart below:



Warehouse Setup

The warehouse set up is shown in the picture below:



This is a 2x2 example of the warehouse. Some important specifications are:

1. There are no boxes on the y-axis.
2. Boxes have a tick leading up to them.
3. Each intersection must go out half the robot's length in distance in each direction.
4. There must be no holes in the grid.

For clarification, a tick refers to a line on the y axis that is used for package locations. A node is an intersection.