PROJECT ALTAIR

Week 1

Hasin Mahtab Alvee
210042174
SWE'21

Contents

Link for Project Altair Git repo

1 Theoretical Part

1.1 Task 01

Theoretical 1. Architecture for a self-driving delivery robot

While a SBC is a full computer with a RAM that can run a complete OS and run all other tasks similar to a computer, the micro-controller can only run specific commands as instructed.

For autonomous driving of a robot, the SBC is a rather safer choice as it can do multiple things at a time, but the micro-controller is a much more efficient choice when it comes to a smaller scale project.

For this scenario, The robot has gear motors and PWM-controlled motor drivers for wheels, servo motors for steering, a rotary encoder, GPS, IMU, and a Stereo Camera. All of these can be brought together with a simple micro-controller, for example - **Arduino UNO**.

So here, the brain will be an Arduino, which can communicate with all the other components. The GPS will start by getting the current location and the target location to set a track for the car. The Motors will be controlled by the PWM motor drivers. These drivers will send the motors to be turned on or off based on the current track obstacle. If there is no obstacle the motors will run on full speed. And to determine the obstacles, we can use the stereo camera. It can sense the obstacles to it's right sensor or left sensor.

- If there is an obstacle to it's right - near, it can slow down, reverse to right and move to left.
- If there is an obstacle to it's left - near, it can slow down, reverse to left and move to right.
- If there's an obstacle to the right - far, it can start to move left.
- If there's an obstacle to the left - far, it can start to move right.
- If there are obstacles to both side - far slow down and stop when they are near.

A rotatory encoder can take care of the turning left and right with inputs from the sensor data and then the IMU can keep the car from losing it's balance while turning or reversing from the data by slowing down the motors or speeding them up.

The reason to pick an Micro-controller over the SBC is for more efficient alogorithms and less complexity. For a bigger scale project the SBC would be safer choice.

References

1. SBC vs Micro-Controllers
2. Functional Architecture for Autonomous Driving
3. Realization of Self Driving Vehicle with Microcontroller

1.2 Task 02

Theoretical 2. Protocol to establish communication between Arduino Mega and Nano

Asynchronous Transmission: Data is sent in form of byte or character. It depends of the amount of time needed to be specified for per byte, which is generally specified as 9600bauds/sec. This transmission is the half-duplex type transmission. In this transmission start bits and stop bits are added with data. It does not require synchronization. Such as UART Communication.

Synchronous Transmission: Data is sent in form of blocks or frames. This transmission is the full-duplex type. Between sender and receiver, synchronization is compulsory. Such as I2C and SPI Communications.

Synchronous communication is more efficient and more reliable than asynchronous transmission to transfer a large amount of data. It is also slower than Synchronous communication. Even though, the UART communication is very much simpler to connect both NANO and MEGA on a PCB, it still falls far behind synchronous communications when it comes to transferring large ammounts of data over limited time.

Between the two types of Synchronous communication methods, the I2C and the SPI, The SPI being a Full-Duplex connection, it can handle data travelling from both the master and the slave at the same time. It is doesn't require a slave id like the I2C, which also makes it so much more faster and can transfer more data. The SPI can work with multiple devices at the same time with the same clock, MISO, MOSI connections. Even though the SPI is only capable of keeping a strong connection within the range of 20cm, whereas the I2C can maintain even at a 1 meters distance, this is where the PCB comes in handy.

Connecting the Arduino NANO and MEGA onto a PCB with a SPI communication is viable as it does not require much distance between the two micro-controllers. As both NANO and MEGA have support for SPI, and has the SPI library to simplify the use, it is a very viable option to use SPI connection for connecting the two devices.

References

1. Synchronous vs Asynchronous Communication
2. Arduino UNO vs NANO vs MEGA
3. Arduino UART Communication
4. Arduino SPI Communication

1.3 Task 03

Theoretical 3.

1.3.1 A. Differnt Motors

Stepper Motors convert a pulsed digital input signal into a discrete mechanical movement. It does not rotate in a continuous fashion and moves in discrete steps or increments. A Stepper Motor is particularly well suited to applications that require accurate positioning and repeatability with a fast response to starting, stopping, reversing, speed control, high holding torque, and lower acceleration. For example, The stepper motor mostly used in 3D printers, CNC machines, medical imagery machinery, printers, security cameras, and other precisely controlled applications.

DC motors have been around since the 1830s and so have been used in a variety of applications. The DC motor, due to its improved working characteristics is still best used in applications requiring constant torque across the motor's speed range. It is fast and continuous rotation motors mainly used for anything that needs to rotate at a high rotation per minute (RPM), and examples are; fans being used in computers for cooling or car wheels controlled by radio, phone vibrators, power tools, car windows, cranes, conveyors, and many more applications that prioritize both sustained output power and price.

The stepper motor is so easy to use, but require some form of a microcontroller to help synchronize their rotor from one pole to the next. The DC motor requires an input voltage to its two leads and does not require external inputs for operation.

References

1. Difference between Stepper motors and DC motors

1.3.2 B. Different Motor driver IC

Motor drivers are used to amplify the voltage and current output from the microcontrollers to the motors, so the motors can be used to it's full potential. Motor drivers use the H-Bridge circuit, which consists of 4 switches, s1,s3 make the motors turn clock-wise and s2,s4 makes it turn anti-clock-wise.

The most common motor drivers are L293D, L298N, TB6612FNG, VNH2SP30.

The L293D is Transistor Based, designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, DC and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications. The L293D is a 16-pin Motor Driver IC which can control up to two DC motors simultaneously, in any direction.

The L298N is a Transistor Based dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 46V, with a peak current up to 3A. It has a built in 5v regulator and supports PWM speed control upto 40kHz. And comes with built in heat sink.

The TB6612FNG is MOSFET based dual channel H-bridge motor drive that can run two separate motors in different direction. Motor voltage ranges from 2.5v to 13.5v, with a peak current of 3.2A. What makes it unique is it had a low power standby mode which makes it very efficient in certain use cases. It also creates the least amount of heat among the drivers and has a built in thermal shutdown feature.

The VNH2SP30 is a MOSFET based single channel motor driver. It can only run one motor at a time, but this can produce high amount of current. Motor voltage being 12 to 16v, it can handle upto 30A at peak. Has an analog built in current sensor. This also produces very low amount of heat even at the high current.

References

1. Why do we use L293D?
2. L298N Driver

1.3.3 C. Using PWM to control motor speed

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital signals are controlled using squares which is why we can only either use high or low speed for motors using digitalWrite. But with analog signals we can use specific values from 0 to 255. As analog signals are like waves, they can be specified and then converted to digital signal for the motors to have controllable speed.

References

1. Motor speed and direction control
2. Basics of PWM

1.3.4 D. Using encoder for feedback

An Odometry system will help the robots to move along the absolute positioning of the field. It uses position wheels which are implemented using optical shaft encoder, which keep the robot from fixing the errors but the down side being it needs to do complex calculation in a short amount of time, but using the ESP32 may solve that problem.

Encoders are typically used for “infinite rotation” applications, such as a drive wheel. The Optical Shaft encoder is used to measure both relative position of and rotational distance traveled by a shaft. It works by shining light onto the edge of a disk outfitted with evenly spaced slits around the circumference. As the disk spins, light passes through the slits and is blocked by the opaque spaces between the slits. The encoder then detects how many slits have had light shine through, and in which direction the disk is spinning.

- Measure Angular Travel
- Determine Rotational Direction
- Calculate Shaft Speed
- Calculate Distance Traveled
- Increase Navigational Control
- More Autonomous Functionality

References

1. Optical Shaft Encoder

1.4 Task 04

Theoretical 4. Arduino Mega vs ESP32 vs STM32

The Arduino MEGA being the fastest in the Arduino family, but when it is compared to the other two at question, It is not as fast as it may seem.

In a simple test to iterate up to 1 million while also lighting a LED, the MEGA falls behind by a lot, it takes almost 7200ms. Whereas, the STM32 and ESP32 both respectively took around 960ms and 160ms. So now let's compare between these two of which can be best used to utilize all the requirements for the projects.

ESP32 being a dual core, is much faster than the STM32, and it has Motor PWM for controlling the multiple motors in any project. ESP32 can perform as a complete standalone system or as a slave device to a host MCU, reducing communication stack overhead on the main application processor. With it's Hybrid Wi-Fi and Bluetooth Chip, it can communicate upto 200 meters in range. For a remote control drone controller it has a Infrared remote controller (TX/RX, up to 8 channels). It has Hall effect sensor, which can be used for the home automation. Finally the Ultra-low-power (ULP) co-processor can use power efficiently.

Which means choosing a ESP32 is better than a STM32.

References

1. ESP32 vs STM32
2. ESP32
3. STM32
4. Arduino MEGA

1.5 Task 05

Theoretical 5. Sensors for a Martian food delivery robot

As sensors play a vital role in any automation driving models, the Mars food delivery car will also need to utilize all sorts of sensors for locationization and positioning.

The depth camera is a better choice for depth mapping, because that's not really what telephoto lenses are intended to accomplish. When you want to map a small segment of an overall object space, but most of the applications for depth mapping — such as autonomous vehicles, as one example.

A 6 Axis IMU is built on top of a Gyroscope and Accelerometer. 9 axis may sometimes have magnetometer. IMU measures and reports a body's specific force, angular rate, and sometimes the orientation of the body using the before mentioned components. So to know the state of the robot on the uneven ground of mars, this is a must.

A LIDAR, Light detection and ranging - is also very important as it measures the distance from a specific object by sending light and catching the reflects in the sensor to measure the distance from the object.

An Odometry system will help the robots to move along the absolute positioning of the field. It uses position wheels which are implemented using optical shaft encoder, which keep the robot from fixing the errors but the down side being it needs to do complex calculation in a short amount of time, but using the ESP32 may solve that problem.

A sonar sensor can sense the movements of objects that might not be in the camera or depth camera sight. So to evade incoming collision from the blind spots, a sonar can be really helpful.

References

1. Depth Camera
2. IMU

1.6 Task 06

Theoretical 6. Maintenance task to be carried out

There are a total of 10 objectives to be done in the Maintenance Task. And the final goal is to be able to complete these objectives. For each task, an operation mode must be specified as well as the operation plan. Tasks can be done either in Manual mode or Autonomous mode.

For the manual mode, A teamviwer will be connected to the ARM computer which will have the controller software running. As for the autonomous mode, the software must be provided as a docker image beforehand.

- **Marker inspection :** This is only for the autonomous mode. We need to provide the scanner will sufficient data about it's surroundings and need to specify it to scan the robot cell to obtain its location along with all components and where what is.
- **Actuation of buttons :** From 9 buttons, the bot must press 4 buttons in a specific order. We'll do this in autonomous mode, and specify the buttons as tags, in the format of strings, for example if button 4-5-7-9 need to be pressed in that order, then we specify (4,5,7,9).
- **IMU module :** This will be done in the manual mode, where we will keep the 3 cameras on and we will pick up the IMU module using the grippers, first we will open the gripper and once IMU is in place we will semi-close and then close the gripper. After that, we will use the joysticks to take the arm to the right position and then semi close and then open the gripper to put the IMU module to the the left panel in a correct orientation. For the orientation, we need to change to rotation mode and we have the Axis panel in the remote, we need to specify the Correct Y-Axis orientation to set up the IMU module in. And this will complete both task 3 and 4 as long as the gripper doesn't touch anything else for at least 3 seconds.
- **Panel opening :** This will be done in the manual mode, First we need to inspect where the Panel is and then using the joystick and gripper we will pick up the lid.
- **Panel cover storage :** This will be done in the manual mode, We need to put the lid to a designated area, and we need to be careful that we don't drop it from any height more of 10cm.
- **Panel inspection :** This will be done in Autonomous mode, where the hidden ID number for the identification will be printed inside the docker image and the arm will use the scanners and then compare the id with the in-built id given to identify the panel.
- **Panel closing :** This will be done in manual mode. Once the panel is detected and identified we use the joy stick and gripper to put the lid at the right window.
- **Secret button :** This will be done in autonomous mode, so an int type of parameter must be given in the docker image so that the arm can identify the secret button and then press it.

- **Home position :** This will be done in Autonomous mode, as we already have the starting position of the arm stored from the first scan, we will reuse that data in the docker image and then the arm will go back to it home position.

2 Logical Part

2.1 Task 01

Logical 1. Help Spirit!!

2.1.1 A. Justify if path Available

There are exactly 7 locations that 'Spirit' is planning on visiting[vertices]. A two-dimensional matrix is given, on which location is directly connected to another location [Edges].

Vertices = 7

Edges = [[0, 1], [1, 2], [2, 0], [3, 4], [4, 5], [5, 6], [6, 3]]

As, the dataset suggests, there is to direct edge or connected path between the 2 and 3 verticies, which means 0,1,2 create a circular graph and 3,4,5,6 create a circular graph. 2 separate circular graphs can not have a common path that would allow spirit to visit all of the vertices if it started on either side of the edges, here I have taken 0 as the starting point to demonstrate the problem. I have not used any graph algorithms, but i have taken inspiration from the DSU Algorithm to come up with this method.

```
vertices = {
    0: True,
    1: False,
    2: False,
    3: False,
    4: False,
    5: False,
    6: False
}

edges = [[0, 1], [1, 2], [2, 0], [3, 4], [4, 5], [5, 6], [6, 3]]
```

If we visit a vertex then the value of that specific vertext should be True, as it is visited. Here, 0 is initially set True as it is the starting point, so it is already visited and all other vetices are False, as not visited yet.

```
for edge in edges:

    if edge[0] in vertices and edge[1] in vertices:

        if vertices[edge[0]] is True:
            vertices[edge[1]] = True
```

Now if, one of the vertex in an array of the edges is true, then the other can also be true, as being in the same array means they have a direct edge between them, so they both can be visited, thus changing the values of the other vertex to True if the prior is True.

```

print(vertices)
```

```
all_values_true = all(vertices.values())


if all_values_true is True:
    print("true")
else:
    print("false")
```

Now if all the vertex have their value changed to true, it means all of the vertices have been visited, thus there is a valid path for Spirit to visit all of them. So, we check the values of all the individual vertex and if all of them are true, then we finally print true, otherwise false.

References

1. Disjoint Set Union (DSU)
2. Find if Path Exists in Graph

2.1.2 B. Diagram for edges

For the given edges

edges = [[0, 1], [1, 2], [2, 0], [3, 4], [4, 5], [5, 3], [6, 7], [7, 8], [8, 6]]

It creates 3 circular separate graphs. One between the 0,1,2 vertices, one between the 3,4,5 vertices and one between the 6,7,8 vertices.

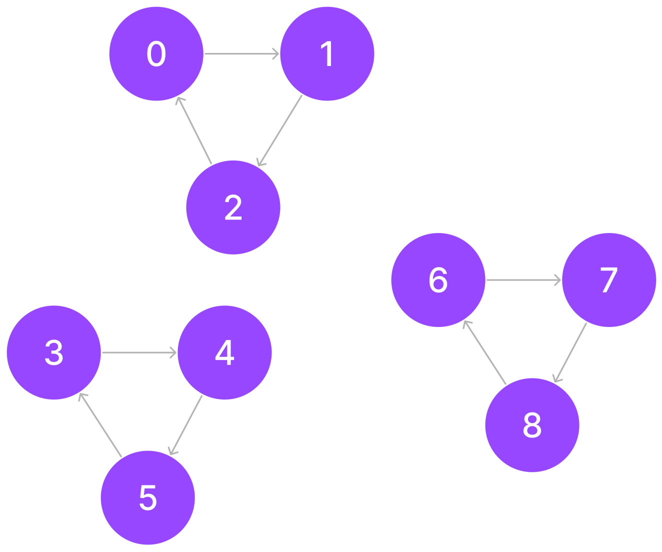

Figure 1: Graph for the given Edges

2.2 Task 02

Logical 2. Brief overview of Shortest-path Algorithms

Dijkstra's Algorithm, can find the shortest path between nodes in a graph. If there is a starting node and and ending node, the Dijkstra Algorithm is used to find the shortest path considering the condition that the path costs the least weight on each edge.

The algorithm keeps track of the currently known shortest distance from each node to the starting node and it updates these values if it finds a shorter path. Once the algorithm has found the shortest path between the starting node and ending node, that node is marked as "visited" and added to the path. The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.

The downside to this algorithm is, it checks all available paths before deciding which is the shortest. Which may not be much of an issue when finding path in small graphs, but when we talking about the large scale, for example US roads, this may take longer than we think.

A* Algorithm, is built on top of the Dijkstra Algorithm, as it also uses the almost same method to keep track of the starting node and the shortest node from it. But it takes into consideration another condition, which is the total distance of the end point from the start in a specific path.

The A* algorithm measures both the weight of edges and the distance, that way some vertices are never traversed through, and it reduces the amount of time needed to find the shortest possible path from all the nodes. This overcomes the shortcomings of the Dijkstra Algorithm with the massive graph sizes.

The downside is, This algorithm is complete if the branching factor is finite and every action has fixed cost. The performance of A* search is dependant on accuracy of heuristic algorithm used to compute the function h(n).

But, both the Dijkstra Algorithm and the A* algorithm don't work with negative edges. They may work, but might or might not give the correct answers. That is when we must use Bellman Ford Algorithm, as it works will negative edges.

Dynamic Programming is used in the **Bellman-Ford algorithm**. It begins with a starting vertex and calculates the distances between other vertices that a single edge can reach. It then searches for a path with two edges, and so on. The Bellman-Ford algorithm uses the bottom-up approach. Bellman Ford Algorithm works on the principles of relaxation. Which basically means, if a pair of vertices cost less than the next vertex, than change the cost with the weight of the next vertex.

Most used in examining a graph for the presence of negative weight cycles, using negative weights, find the shortest path in a graph, routing is a concept used in data networks.

The **Floyd Warshall Algorithm** is for solving all pairs of shortest-path problems. The problem is to find the shortest distances between every pair of vertices in a given edge-weighted directed Graph. And it also follows the dynamic programming approach.

It works by means of keeping a matrix of distances between each pair of vertices and updating this matrix iteratively till the shortest paths are discovered. Mostly used in Routing algorithms, Airline networks, Computer networks.

Since it works with the square of n, it can get really big in terms of matrix when a bigger graph is in question. It can be harder to understand than some other simpler algorithms.

References

1. Dijkstra Algorithm
2. Bellman Ford Algorithm
3. Bellman Ford Algorithm
4. Floyd Warshall Algorithm
5. Floyd Warshall Algorithm

3 Microcontroller Part

3.1 Task 01

Practical 1. I2C Communication between two Arduino UNO

The simplest and efficient way to connect two Arduino UNO is the I2C communication. In this communication, we have a Master and a Slave. The slave is indentified with a slave id, and the 4 data pass types are, SDA, which is the line that sends and recieves data between master and slave, which is usually A4 in UNO and SCL, which is the clock is usually A5.

MicroController Task 01 TinkerCad Link

Here in the Master side code for the Arduino, we first need to include the built-in arduino library for the I2C communication, Wire.h, and need to initialize it with the wire.begin() method.

```
#include <Wire.h>

void setup() {
  //Initialize I2C
  Wire.begin();
}
```

Setting a vairable x to any value and then send it to the slave.

```
byte x = 27;

void loop() {

  //Define the address where we want to connect to slave
  Wire.beginTransmission(12);
  //Send a string
  Wire.write("Master is Sending ");
  //sending the value of x
  Wire.write(x);
  //Ending the single transmission for next loop
  Wire.endTransmission();

  //Increment by 3
  x = x+3;
  delay(1000);
}
```

The transmission starts with the beginTransmission(12) method on the specified address 12. Then data is passed, in this case data is the string and then the x variable which is incremented after each transmission ends.

On the slave side, the setup starts with the same address 12, to which the master send the data.

```
#include <Wire.h>

void setup() {
  //Starting connection as the same address as Master
  Wire.begin(12);
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);
}
```

Here a specific recieveEvent is triggered at the very beginning so that the first data can be read.

```
void loop() {
  delay(100);
}



void receiveEvent(int howMany) {
  while (1 < Wire.available()) {
    char c = Wire.read();
    Serial.print(c);
  }
  //recieveing the x value
  int x = Wire.read();
  Serial.println(x);
}
```

The recieveEvent reads the strings each as a character with a while loop and reads the value of x everytime. Finally prints the result to the Serial monitor, the string and followed by the value incremented by 3.

```
Master is Sending 27
Master is Sending 30
Master is Sending 33
Master is Sending 36
Master is Sending 39
Master is Sending 42
```

References

1. I2C Communication

3.2 Task 02

Practical 2. Controlling a motor with Arduino

Rotatory Encoders use a magnetic or optical sensor to measure the angular movement of the motors and determine the position. It is used to control the speed of motors. A common type of encoding is Incremental. It depends on two pulsed outputs to calculate the position and direction of the motor. If the top set precedes the bottom, it indicates a clockwise rotation, and if vise-versa, then it indicates an Anti-clockwise rotation. A sample TinkerCad example for controlling a motor speed with a rotatory encoder is given here :

MicroController Task 02 TinkerCad Link

Here, since I am using a H-Bridge motor driver, I used the Enable pins as the PWM pin, but I could not make it work properly that way, but if I could use a PWM motor driver, for example a Cytron MD10C PWM motor driver, this might have worked.

The concept is very simple, we connect the Arduino 10 as the PWM pin and 12 as the direction pin. We take pulses as input from the pin 3 excluding all noises. We set variables as intervals to calculate per pulse, and encoder value.

```
#define ENC_COUNT_REV 100
#define ENC_IN 3
#define PWM 10
#define DIR 12


int speedcontrol = 0;
volatile long encoderValue = 0;
int interval = 1000;
long previousMillis = 0;
long currentMillis = 0;
int rpm = 0;
int motorPwm = 0;
```

Setting the PWM and Direction pins as output, so that the motor speed and direction can be controlled. Using the attachInterrupt to read digital value from the input pin 3 and calling the updateEncoder method, which increments the value for each pulse from the encoder.

```
void setup()
{
  Serial.begin(9600);

  pinMode(ENC_IN, INPUT_PULLUP);

  pinMode(PWM, OUTPUT);
  pinMode(DIR, OUTPUT);

  attachInterrupt(digitalPinToInterrupt(ENC_IN), updateEncoder, RISING);

  previousMillis = millis();
}


void updateEncoder()
{
  // Increment value for each pulse from encoder
  encoderValue++;
}
```

Here, we read the speedControl value from the potentiometer, to make the motor spin, and we using map function to convert the arduino 1024base to 255base signal. And we write the value of the motorPWM to the motor driver. To check if a second has passed, meaning a pulse has passed, we comapre the currentSecond with the previousSecond in the Setup, and if the currentSecond is larger, we change the value of previousSecond with currentSecond. Finally, to calculate RMP, we multiply the encoderValue with 60 for 60 seconds and divide it my the Encoder built it REV value. And we print the values to the serial monitor.

```
void loop()
{

    // Control motor with potentiometer
    motorPwm = map(analogRead(speedcontrol), 0, 1023, 0, 255);

    // Write PWM to controller
    analogWrite(PWM, motorPwm);

  // Update RPM value every second
  currentMillis = millis();
  if (currentMillis - previousMillis > interval) {
    previousMillis = currentMillis;


    // Calculate RPM
    rpm = (float)(encoderValue * 60 / ENC_COUNT_REV);

    // Only update display when there is a reading
    if (motorPwm > 0 || rpm > 0) {
      Serial.print("PWM VALUE: ");
      Serial.print(motorPwm);
      Serial.print('\t');
      Serial.print(" PULSES: ");
      Serial.print(encoderValue);
      Serial.print('\t');
      Serial.print(" SPEED: ");
      Serial.print(rpm);
      Serial.println(" RPM");
    }

    encoderValue = 0;
  }
}
```

References

1. DC Motor with Encoder
2. Using Rotary Encoders with Arduino
3. Rotatory encoder documentation
4. Motor driver tutorial