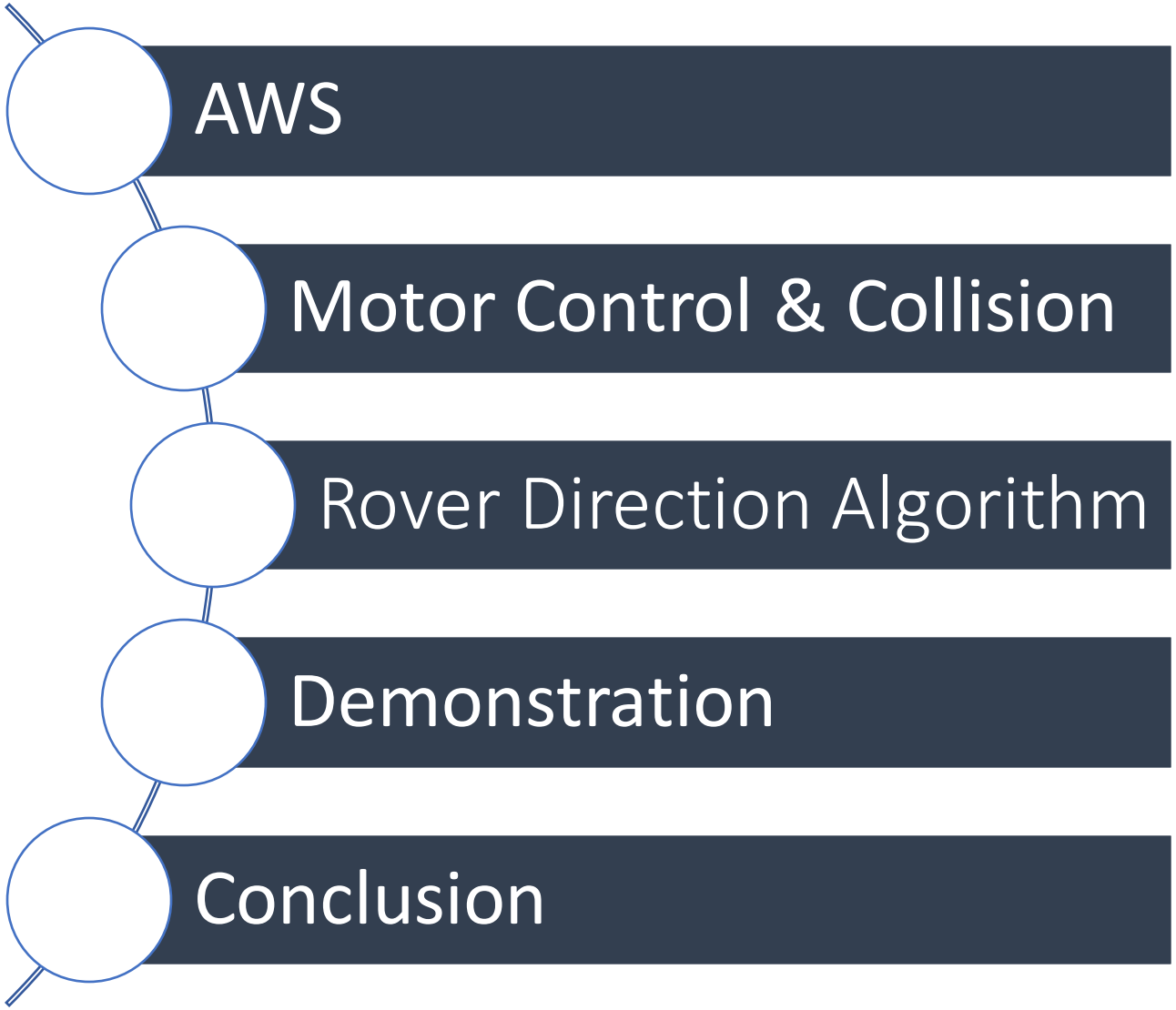
Self-Running Rover

Embedded System Lab

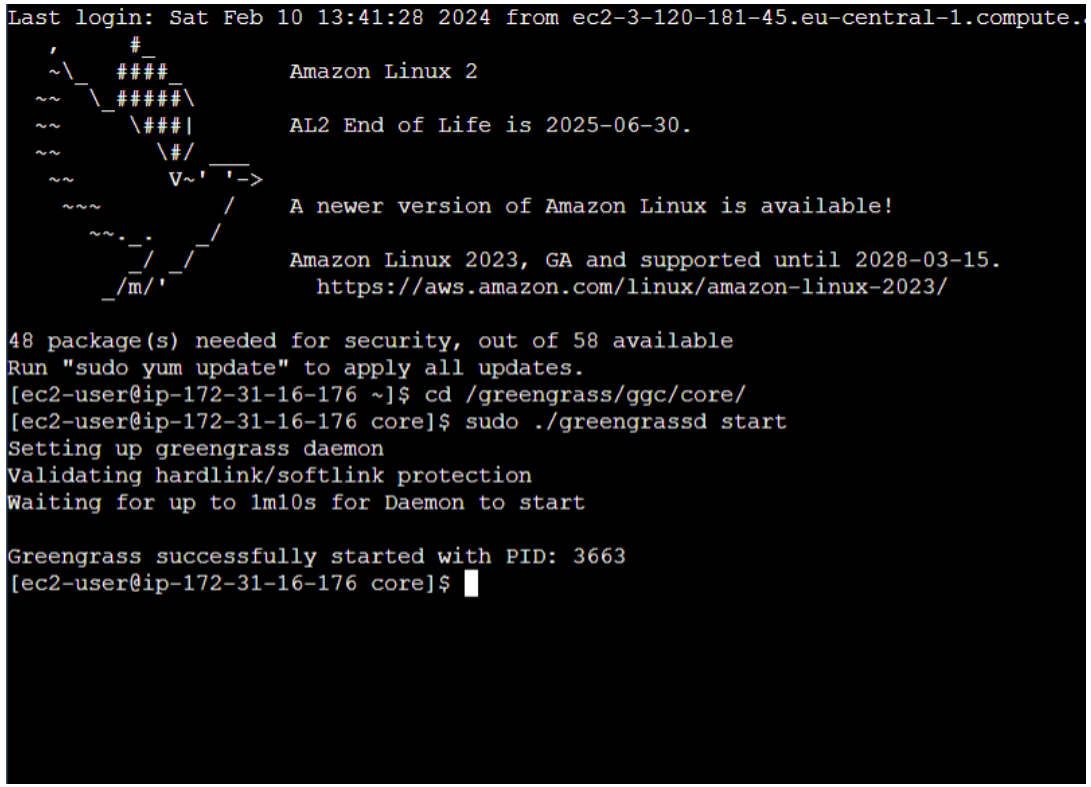
7219288 Nhat Quang Nguyen
7219037 Nhat Lam Nguyen
7219281 Igor Risteski

Outline

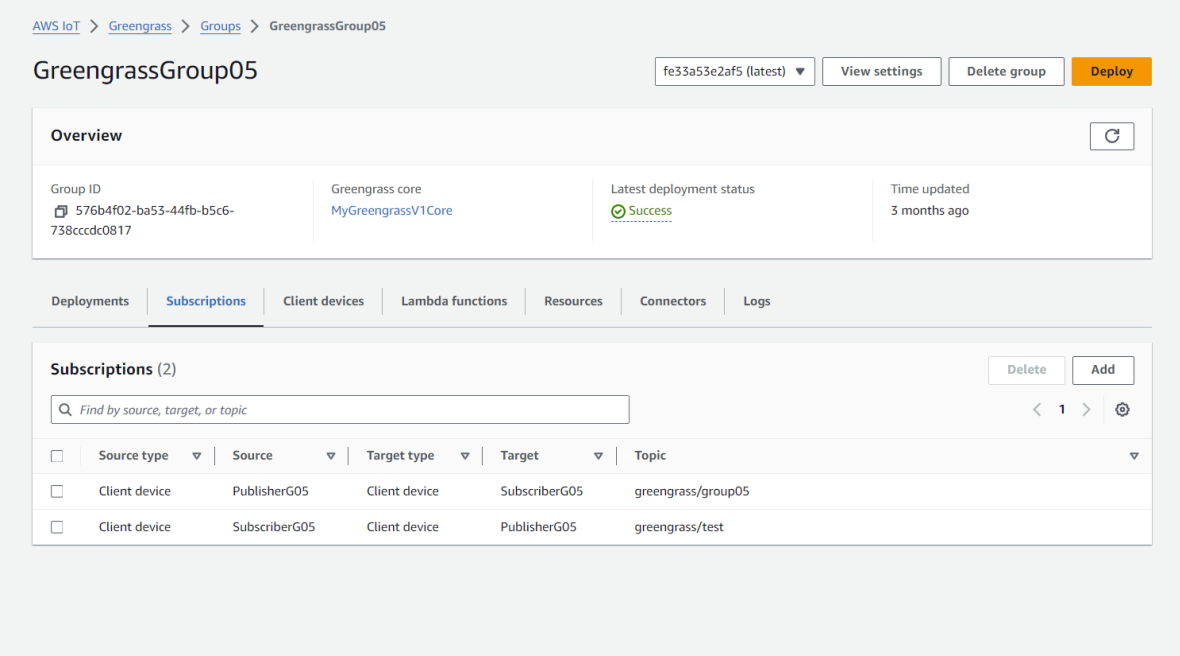

AWS
Motor Control & Collision
Rover Direction Algorithm
Demonstration
Conclusion

AWS

- EC2 Instance



AWS



- Greengrass Group & MQTT Subscriptions

AWS

- Setting up the subscription to the topics from the rover in file AWS.cpp

```
/* The MQTT topics that this device should publish/subscribe to */
#define AWS_IOT_PUBLISH_TOPIC   "greengrass/group05"
#define AWS_IOT_SUBSCRIBE_TOPIC "greengrass/test"
```

```
/* Subscribe to a topic */
client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);
client.subscribe(AWS_IOT_PUBLISH_TOPIC);
```

AWS

```
void messageHandler(String &topic, String &payload) {
  if (topic == "greengrass/group05")
  {
    receivedRoverPayload = payload;
  }
  if (topic == "greengrass/test") {
    receivedTargetPayload = payload;
  }
}

/* Create a message handler */
client.onMessage(messageHandler);
```

Function to receive message from AWS in AWS.cpp

```
void extractCoordinates(const String &payload, int &x, int &y)
{
  int a, b;
  int startIndex = payload.indexOf('(') + 1;
  int commaIndex = payload.indexOf(',', startIndex);
  int endIndex = payload.indexOf(')', commaIndex);

  String xStr = payload.substring(startIndex, commaIndex);
  String yStr = payload.substring(commaIndex + 1, endIndex);

  x = xStr.toInt();
  y = yStr.toInt();
}

extractCoordinates(receivedRoverPayload, roverX, roverY);
extractCoordinates(receivedTargetPayload, targetX, targetY);
```

Decoding Function to transform the Data from AWS to coordinates from the targe and the rover in main.cpp

```
class configureAWS:
        host = 'a1ah0y2qxdql5g-ats.iot.eu-central-1.amazonaws.com' # set here your end-point
        rootCAPath = 'AmazonRootCA1.pem'
        certificatePath = '895ad5c098d5bc9c65b876962226663d6e3ba3e87486a745351fc1934a3a2883-
certificate.pem.crt'  #update to the name of your certificate
        privateKeyPath = '895ad5c098d5bc9c65b876962226663d6e3ba3e87486a745351fc1934a3a2883-
private.pem.key'  #update to the name of your certificate
        clientId =  'SubscriberG05'  # set your thing name
        topic_rover = 'greengrass/group05'
        topic_target = 'greengrass/test'
        #port=8883
        useWebsocket=False
```

AWS

- Configure the AWS Device for the Virtual Machine, which will be sending coordinates data of rover & target to AWS

Motor Control

src > motorDriver.cpp > motor_direction(Motors, Direction)

```
1   #include "Arduino.h"
2   #include "motorDriver.h"
3
4   static uint8_t inA1 = 16;          /* Pin for Motor Right */
5   static uint8_t inA2 = 17;          /* Pin for Motor Right */
6   static uint8_t inB1 = 19;          /* Pin for Motor Left */
7   static uint8_t inB2 = 33;          /* Pin for Motor Left */
8   static int freq = 20000;
9   static uint8_t Channel1 = 0;
10  static uint8_t Channel2 = 1;
11  static uint8_t motorChannel1 = 5;   /* PWM Pin for Motor 1*/
12  static uint8_t motorChannel2 = 18;  /* PWM Pin for Motor 2 */
13  //PWM - Pulse Width Modulation -> technique to adjust digital pulses
14  // to generate different avr dc voltages.
15  static uint8_t resolution = 8;
```

```
void mclass::motor_direction(Motors motor_ch, Direction dir) {

  if (motor_ch == 0)
  {
    if (dir == Forward)
    {
      digitalWrite(inA1,LOW);
      digitalWrite(inA2,HIGH);
    }
    else
    {
      digitalWrite(inA1,HIGH);
      digitalWrite(inA2,LOW);
    }
  }

  else
```

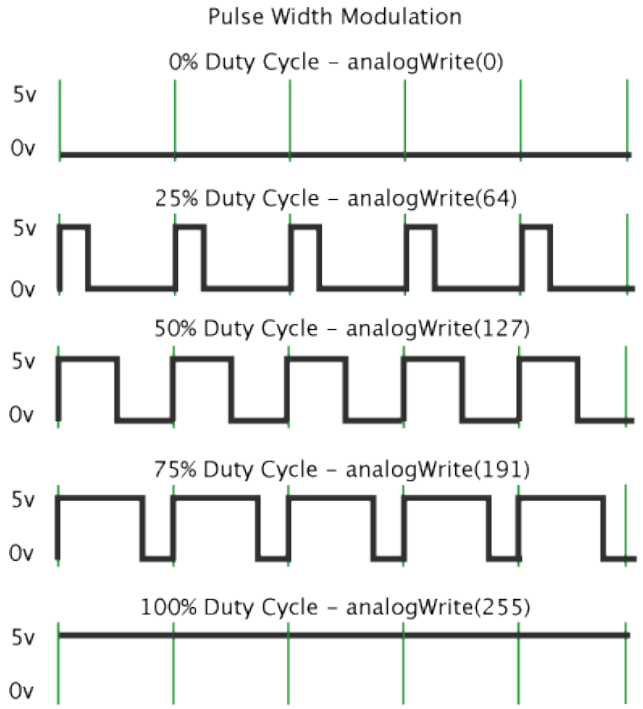
- Include needed libraries
- motorDriver library

-> Set pins

-> use PWM to control motors

-> control motor direction

- Create task for motor control




```
  xTaskCreate(
      taskMotorControl,   /* Task function. */
      "taskMotorControl", /* Name of task. */
      20000,              /* Stack size in bytes. */
      NULL,               /* Parameter passed as input of task */
      1,                  /* Priority of task. */
      NULL);              /* Task handle. */
}
```

Motor Control - main

```
int i = 0;
void taskMotorControl(void *parameter)
{
  for (;;)
  {
  //   // Extract roverX and roverY from receivedRoverPayload
    extractCoordinates(receivedRoverPayload, roverX, roverY);
    extractCoordinates(receivedTargetPayload, targetX, targetY);


    if (previousTargetX != targetX)
    {
      i++;
      motorDriver.set_speed(MotorRight, Forward, roverSpeed + 130);
      motorDriver.set_speed(MotorLeft, Backward, roverSpeed + 130);
      vTaskDelay(2100 / portTICK_PERIOD_MS);
    }
    else
    {
      // Calculate distance between current rover coordinates and target coordinates
      float distance = sqrt(pow(targetX - roverX, 2) + pow(targetY - roverY, 2));
      Serial.println(distance);
      // Calculate angle between current rover coordinates and target coordinates
      float angle = atan2(targetY - roverY, targetX - roverX) * 180 / PI;

      // PID Control
```

- Infinite loop
- Extract coordinates of rover and current target
- Check target and update target num (i)
- Rotate in place OR
- Calculate smallest distance and angle
- Update speeds using PID control
- Call stop function when final destinaton reached 3 times

```
void mclass::motor_all_stop() {

    digitalWrite(inA1,LOW);
    digitalWrite(inA2,LOW);
    digitalWrite(inB1,LOW);
    digitalWrite(inB2,LOW);
    digitalWrite(inB1,LOW);
    digitalWrite(inB2,LOW);
}
```

Collision and obstacle avoidance

```
26  void setup()
27  {
28    // pinMode(LED_BOARD, OUTPUT);
29    Serial.begin(9600);
30
31    motorDriver.SETUP();
32
33    if (collisionSensor.init())
34    {
35      Serial.println("Collision sensor initialized successfully.");
36    }
37    else
38    {
39      Serial.println("Failed to initialize collision sensor.");
40    }
41
42    collisionSensor.setThreshDistance(200); // Set threshold distance to 200 mm
43
44    delay(1000);
45
46    xTaskCreate(
```

- In setup try to initialize collision sensor (OPT301)
- Set threshold distance
- Call collision class

- Check sample and validity
- Check sensor output and compare with threshold distance for all segments
- Update motor speeds to turn more towards one side and avoid obstacle

collision.cpp | main.cpp | motorDriver.cpp

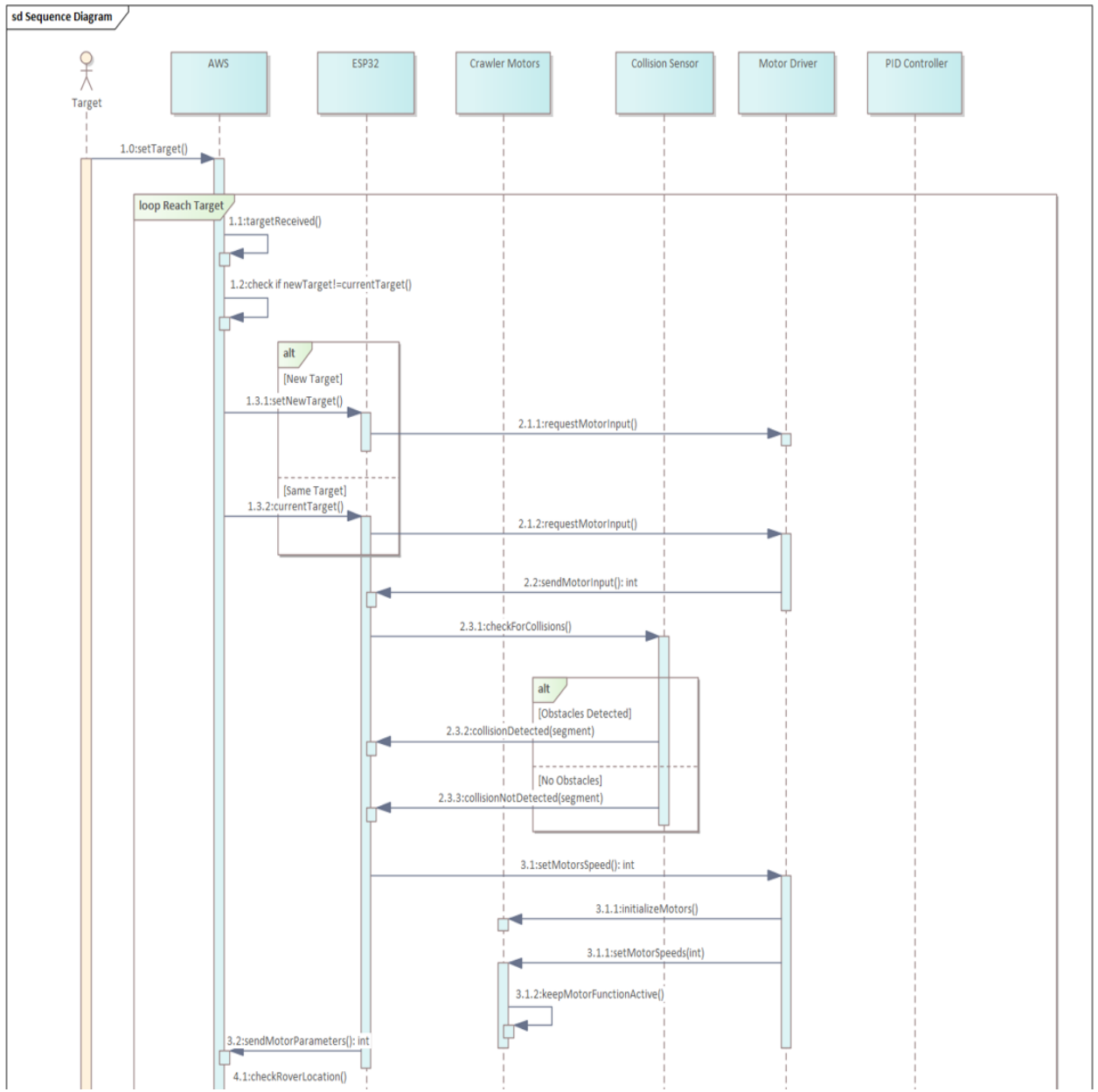
src > collision.cpp > init()

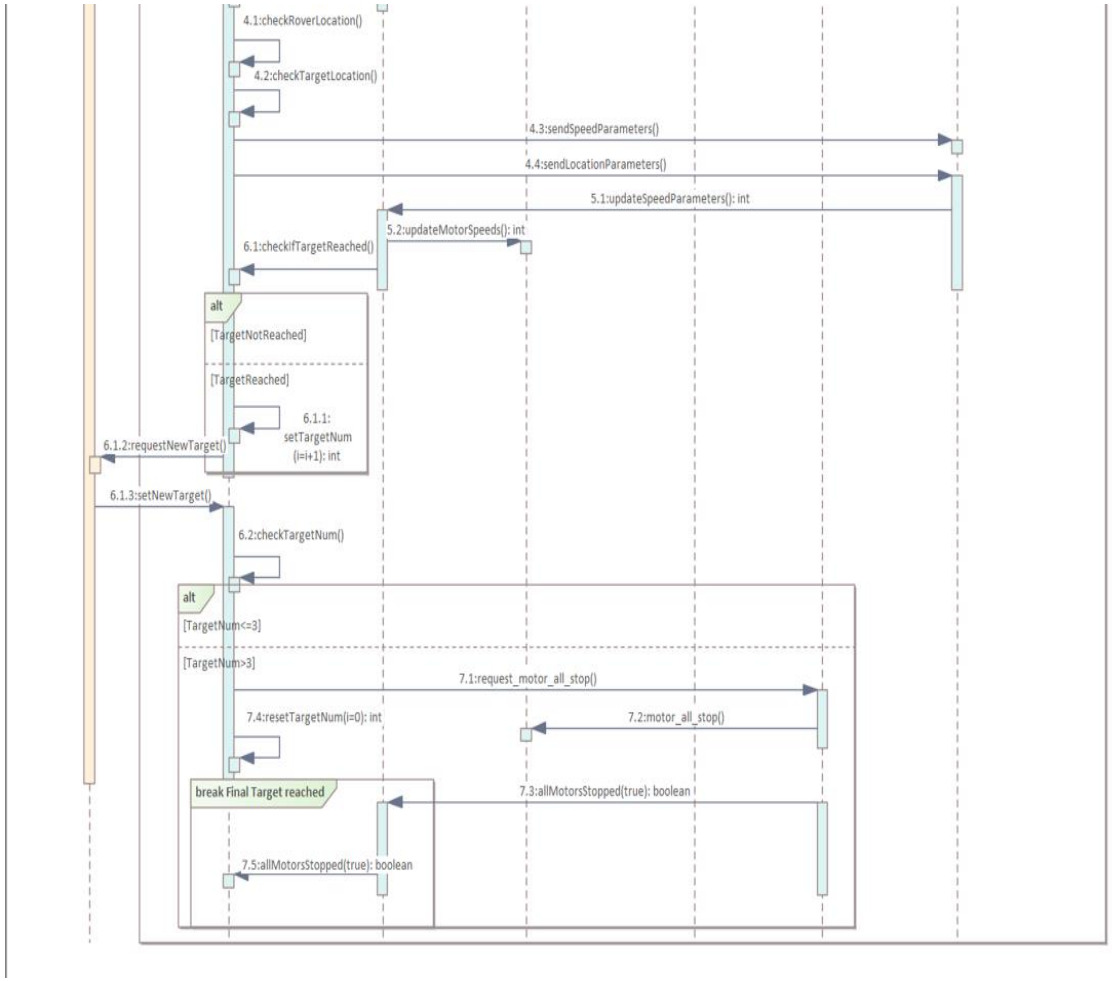
```
1   /**@file collision.cpp
2    * @brief Uses an OPT3101 seg_distance sensor to detect collisions. */
3   #include <Arduino.h>
4   #include "collision.h"
5   #include <Wire.h>
6
7   bool Collision::init()
8   {
9       Wire.begin(27, 26);
10
11      Serial.println("starting");
12
13      /* Wait for the serial port to be opened before printing */
14      /* messages (only applies to boards with native USB) */
15      _sensor.init();
16      if (_sensor.getLastError()) {
17          Serial.print(F("Failed to initialize OPT3101: error "));
18          Serial.println(_sensor.getLastError());
19          return false;
20      }
21      _sensor.setBrightness(OPT3101Brightness::Low);
22      _sensor.setChannel(0);
23      _sensor.startSample();
24      return true;
25  }
```

```
27  Collision::segments Collision::warning()
28  {
29      while (_sensor.isSampleDone() == false)
30          delay(1);
31      _sensor.readOutputRegs();
32
33      enum SEG_NAME { TX0 = 0, TX1, TX2 };
34      enum SEG_NAME cur_seg = (enum SEG_NAME)_sensor.channelUsed;
35      bool valid_sample = (_sensor.amplitude >= _thresh_amplitude) ? true : false;
36
37      int16_t seg_distance = valid_sample ? _sensor.distanceMillimeters : -1;
38      bool warning = false;
39      if (valid_sample && (seg_distance <= _thresh_distance))
40          warning = true;
41
42      _sensor.nextChannel();
43      _sensor.startSample();
44
45      switch (cur_seg) {
46      case TX0:
47          _segments.l = warning;
48          break;
49      case TX1:
50          _segments.c = warning;
51          _c_distance = seg_distance;
52          break;
53      case TX2:
54          _segments.r = warning;
55          break;
56      }
57      return _segments;
```

```
void taskSensorDataGet(void *parameter)
{
  Collision::segments segments;
  for (;;)
  {
    segments = collisionSensor.warning();
    if(segments.c);
    {
      Serial.println("Center collision detected!");
      motorDriver.set_speed(MotorRight, Forward, 100);
      motorDriver.set_speed(MotorLeft, Forward, 200);
      vTaskDelay(3000 / portTICK_PERIOD_MS);
    }
  }
  vTaskDelete(NULL);
}
```

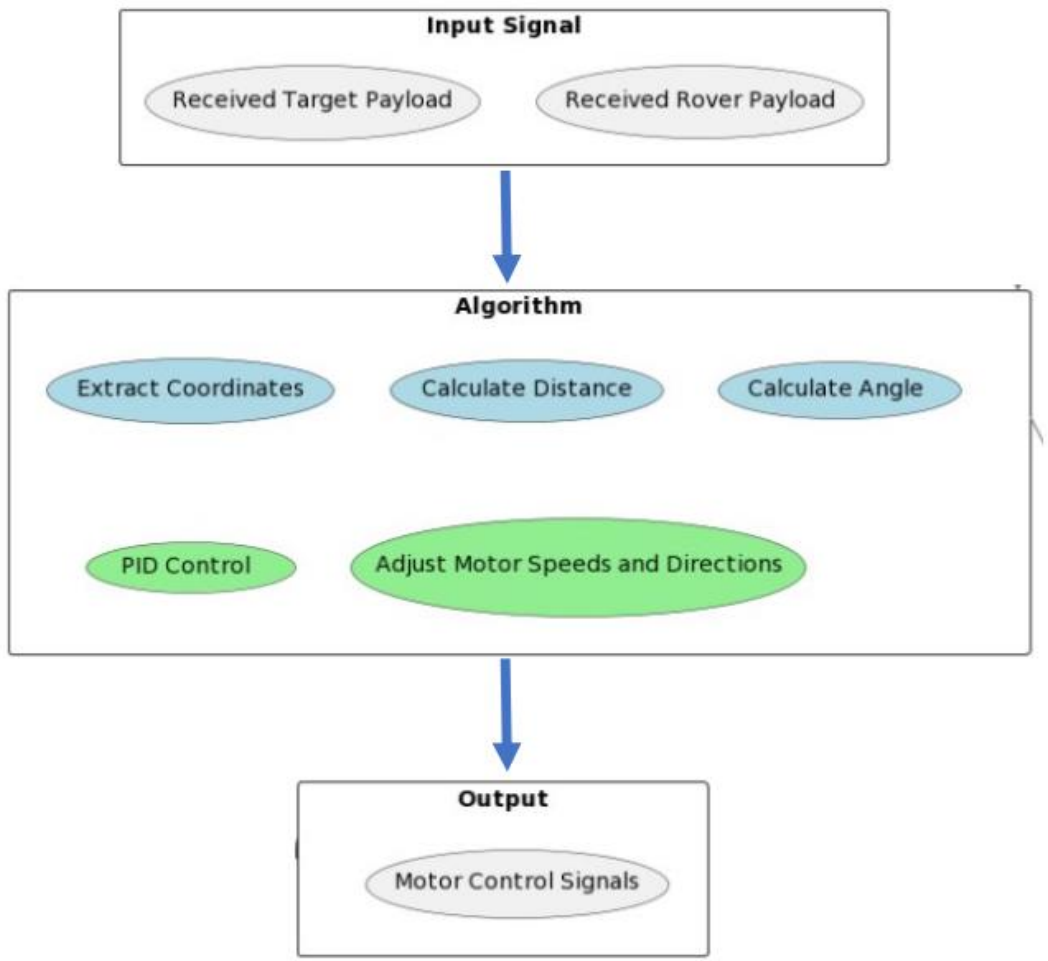
Sequence Diagram

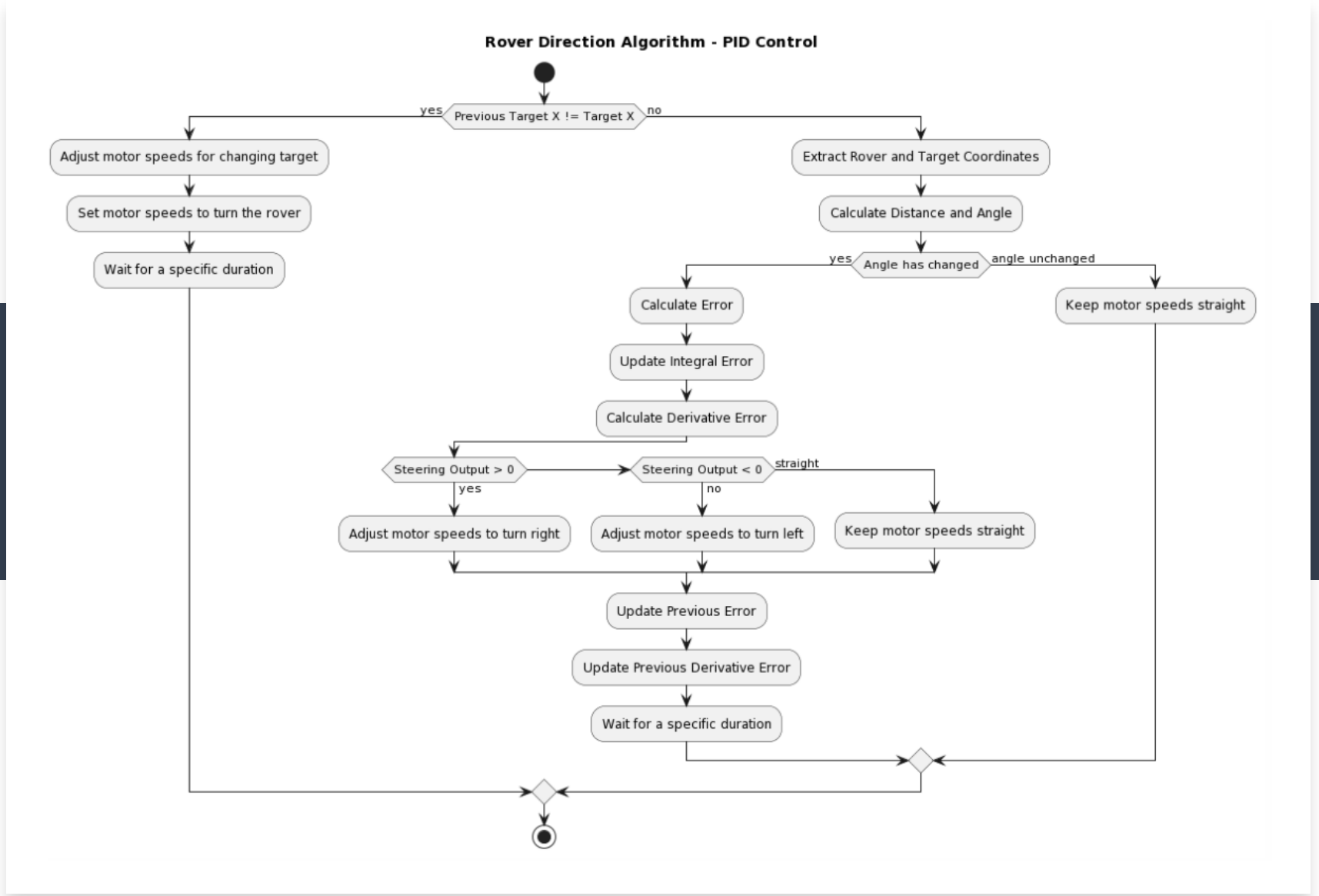




Part 1 ⟵ ⟶ Part 2

Control Algorithm





Rover Direction Algorithm