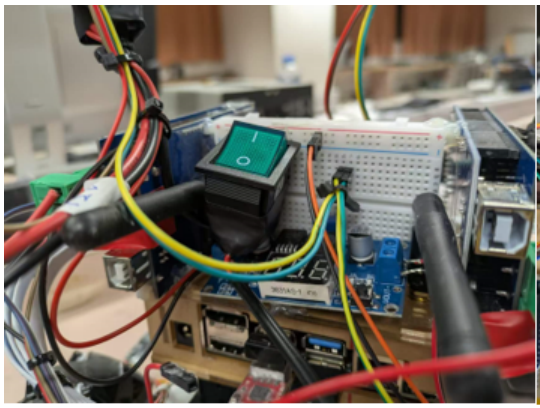
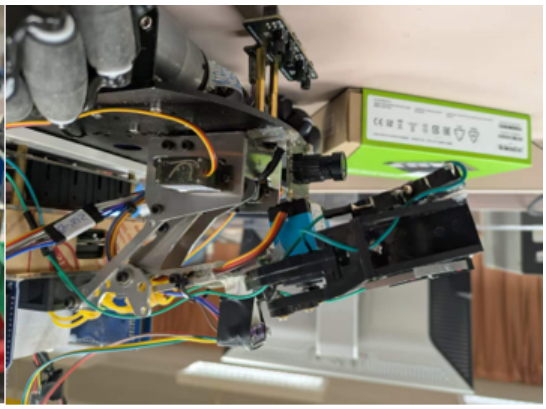
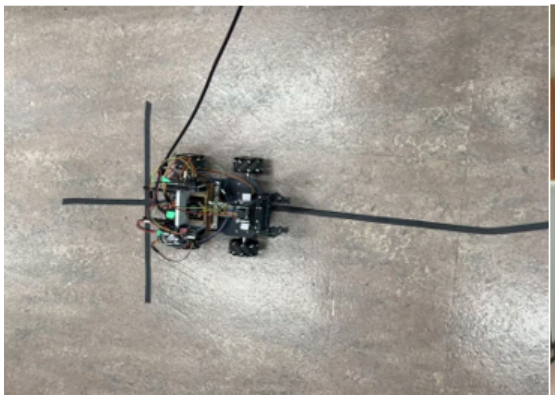
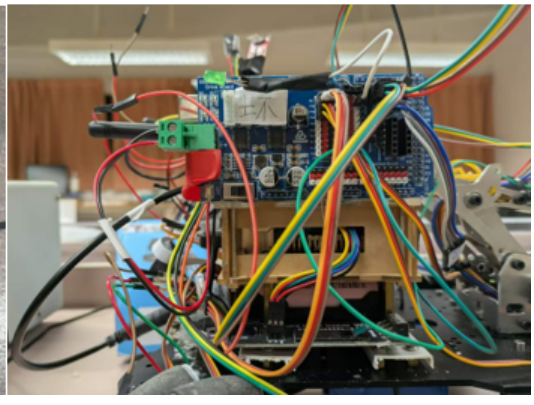
ELEC3848 Final Project Report
Autonomous Towing Tractor for Model Plane Airfields

Group number	D3-4	
Group members	Jin Yushang	3035855064
	Zou Chunyu	3035946629
	Kuang Anke	3035845708
	Ding Jiayi	3036103626

Abstract

In this project we made an autonomous model plane towing truck for model plane airfields. It uses two Arduino mega 2560 boards and one Nvidia Jetson Nano. It can move along traces, detect the plane with camera, automatically move towards the plane, grab the plane with a roboarm, and finally tow the plane back to home.

Arduino code: `https://github.com/EricEricEricJin/ELEC3848-Group-Project-MCU`
Jetson code: `https://github.com/Zou-2004/ELEC3848-Jetson/tree/main/tracker_test_v3`
Demo vodeo: `https://www.youtube.com/watch?v=utF52RqBWJA`

Contents

1 Introduction

Nowadays, many model aircraft need to be routinely tested and evaluated in the model plane airports(e.g., HKMEC air field in Figure 1. Model plane airports, similar to airports designed for real airplanes, also seek to minimize the waiting time for each individual airplane by carefully considering the current time, availability of the runway, and position of each airplane. The delivery of the airplane to a specific position is always an important task for the model plane airport, but it often consumes a lot of time and human resources. Specifically, the model planes need to be placed on the runway before taking off and manually dragged from the runway after landing.

Due to the fact that the process of dragging the model airplane to a predefined location is highly repetitive and time-consuming, it is reasonable to then propose a tractor similar to the tractor in real airports as shown in Figure 2. The tractor in this case should be generally replace the task as performed by human in the past. Thus, in order to accomplish these tasks, there are several goals or requirements of this project: 1. The airplane should be almost automatic, and people only need to predefine the starting point and ending point beforehand. 2. The tractors are able to handle diversified airplanes with different parameters like height. 3. The airplane should be hard enough in material so as to avoid short-time maintenance and even replacement. Under these initiatives, we made our own version of autonomous towing tractor for model airfields. Our car has the following functions: 1. Detect and locate the target airplane in the airfield. 2. Grab the handle of the airplane and release it in the end through robot arm and mechanical claw. 3. Track the light spot as installed right before the landing gear. 4. Send real-time data, such as humidity and pressure, to the airplane.

Figure 1: HKMEC airfield

Figure 2: Tractor for real airplanes

2 Hardware connection

Figure 3 shows the schematic diagram of our robot, power cables are not shown. We made several change to the previous proposal due to some realistic conditions. We first remove UWB and GPS from the previous proposal. This is because we already included enough number of functions, while GPS and UWB will not significantly affect the result of our robot. Secondly, Instead of employing 4 Degree of Freedom(Dof), we finally decided to use three Dofs bacause too many Dofs are not necessary to achieve our tasks. Finally, we used one camera instead of two cameras, and remove the unnecessary microphone.

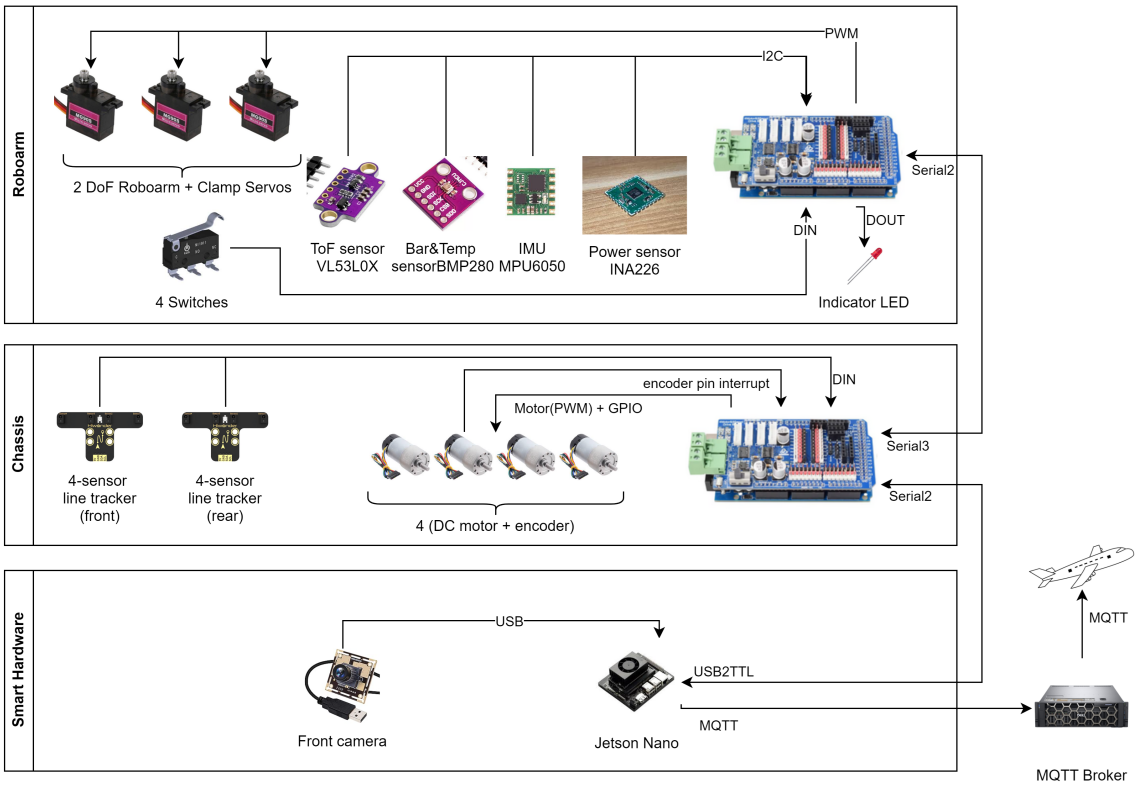


Figure 3: Schematic diagram of the robot. Chassis board is connected to chassis movement related hardware (2 line trackers and 4 DC motors), Roboarm board is connected to roboarm and sensor related hardware (3 PWM servos, micro switches, ToF, BMP, IMU, and INA sensors). The Jetson is only connected to the chassis board through TTL serial and the two Arduino boards are inter-connected with serial port.

3 Program structure

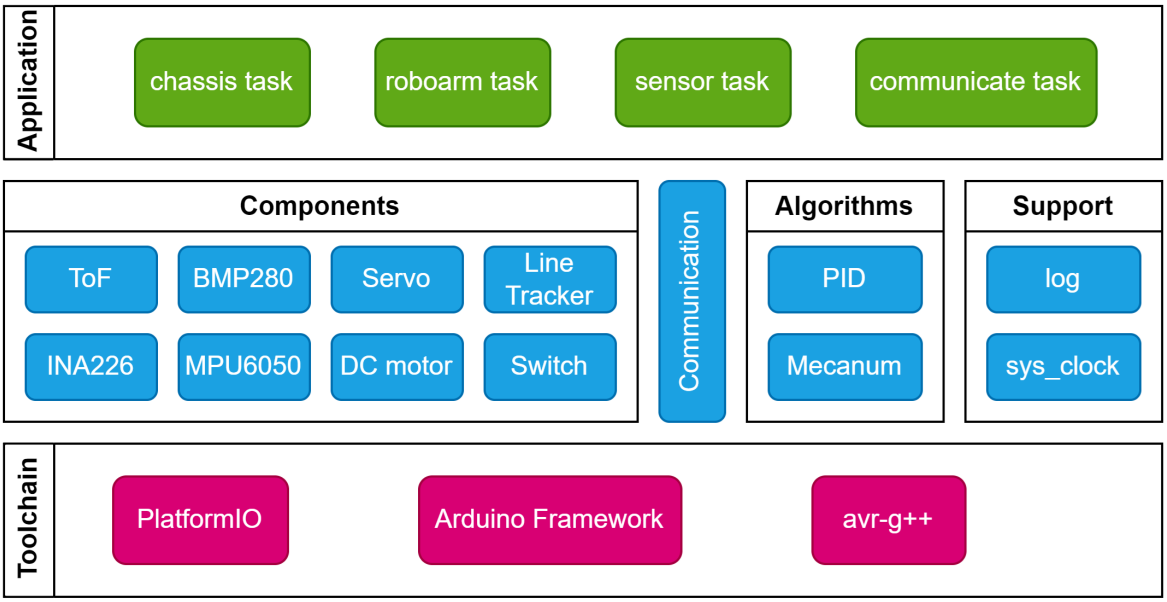


Figure 4: Arduino program structure

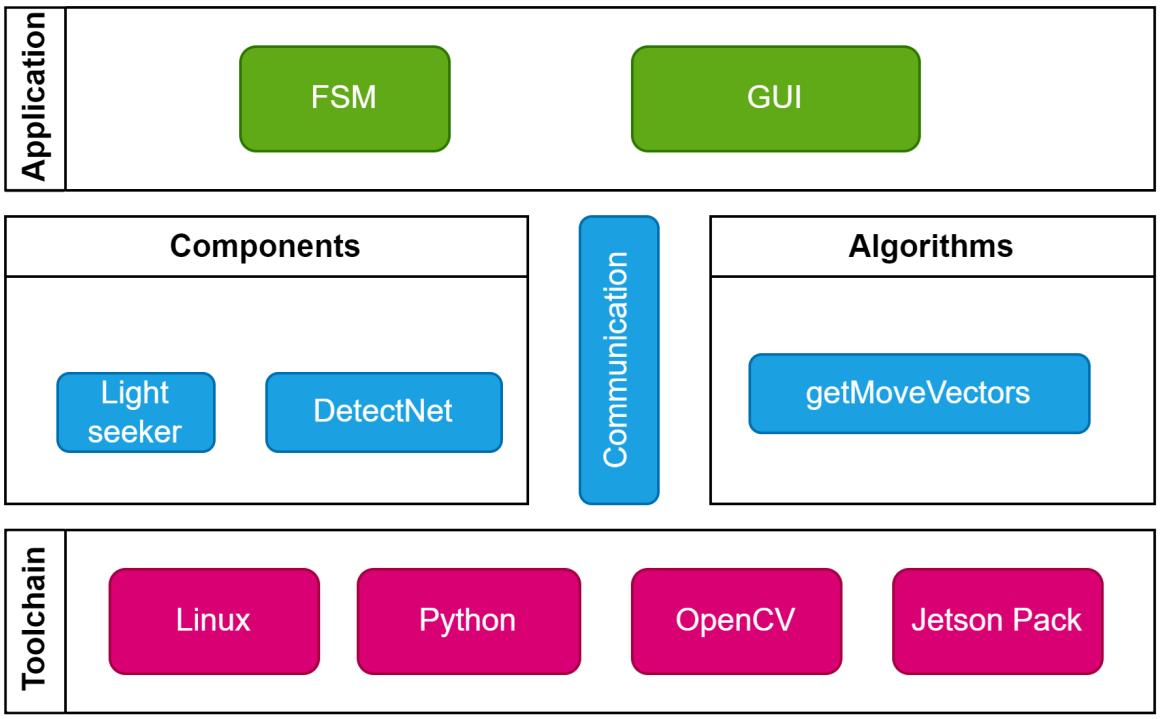


Figure 5: Jetson program structure

4 FSM and Flowcharts of our robot

4.1 Jteson Nano

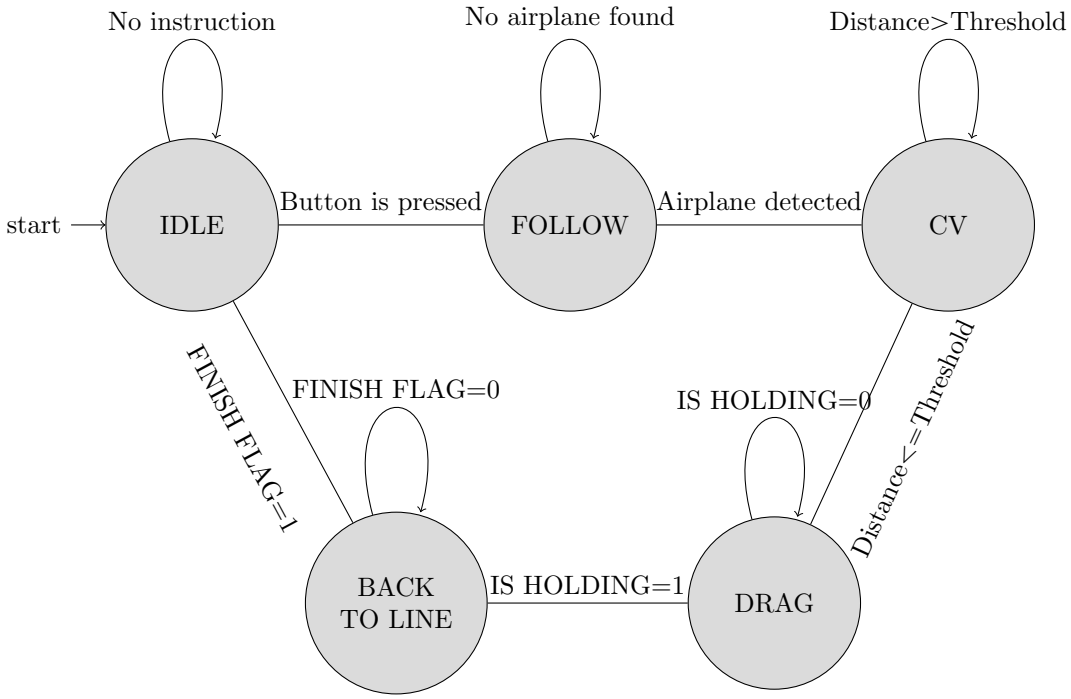


Figure 6: Operation FSM

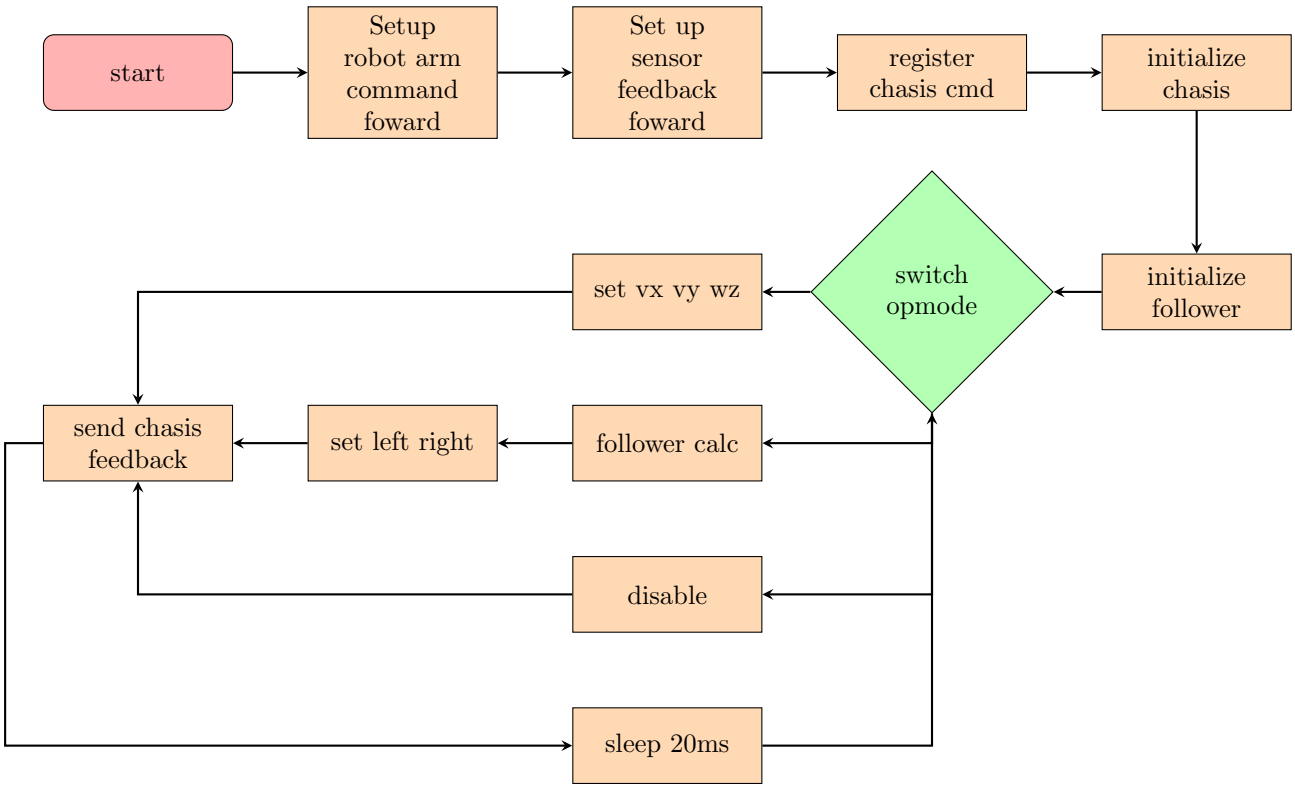
The FSM above illustrate how our robot works.

- Initially, after running the program, the robot will enter the "IDLE" mode. In this case, it will stop moving and wait for coming instructions. The program has a GUI allowing users to start and stop the task by clicking buttons.
- When the Start button is pressed, The robot will enter the "FOLLOW" mode to follow the black line that is placed beforehand. During the "FOLLOW" mode, the robot will also detect the airplane using detectnet.
- When the robot successfully see the airplane, it will enter the "CV" mode. At this state, the robot will move towards the light source on the landing gear of the airplane.
- When the distance between the airplane and the robot is close enough, the robot will enter the "GRAB" mode, the robot arm and the mechanical claw are activated to hold the handle on the airplane. There are in total 4 sensors on the claw with each's NC (normal-close) terminals connected in series, when anyone of them open, the circuit is open and "IS HOLDING" flag will become true, which means that the airplane is successfully dragged.
- When IS HOLDING is true, the robot will enter the "BACK TO LINE" state to go back to the line by moving backwards.
- As the line tracker at the back of the robot detect the black line, it will enter the "HOMING" state to track the line just like the initial "FOLLOW" stage, but in reverse direction. There will be a finish sign in the end of the black line such that all the sensor from the line tracker will detect the black color.

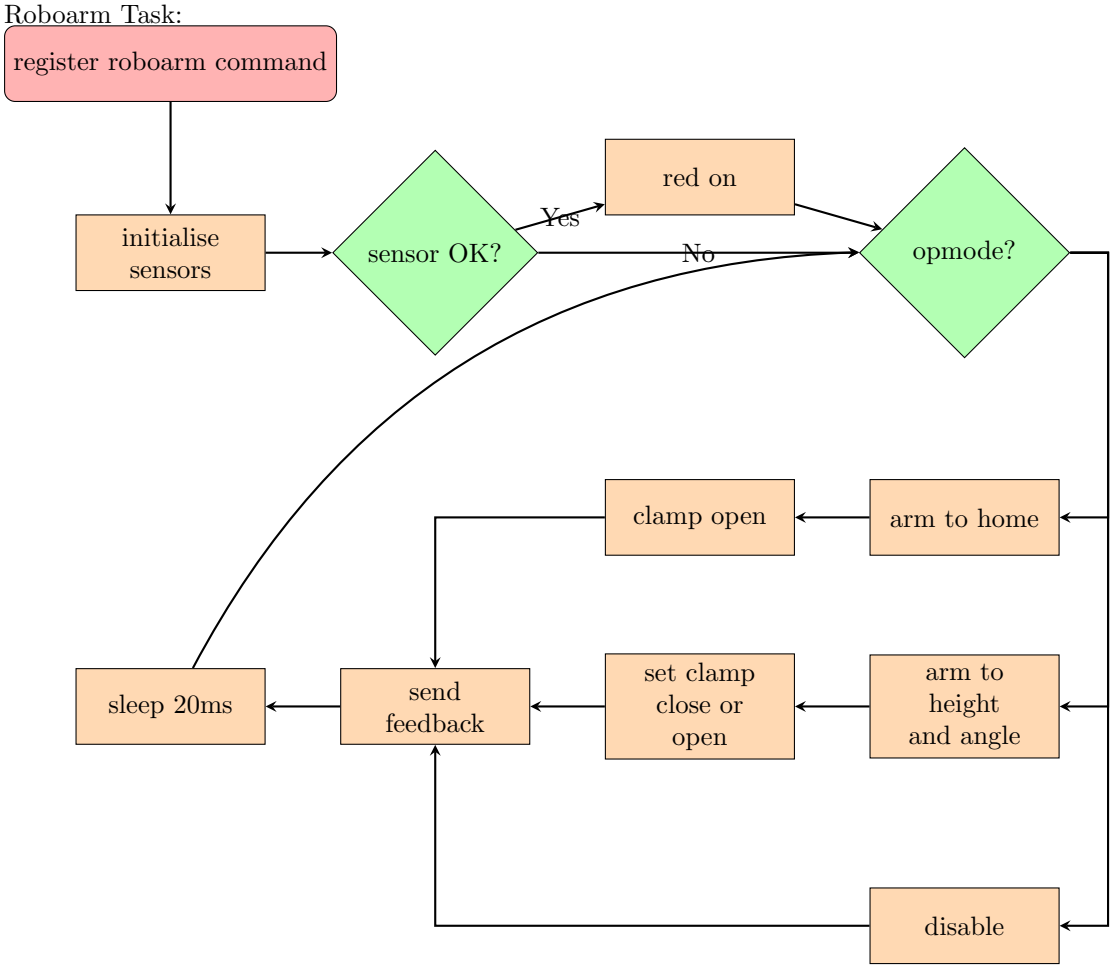
In this case, the finish flag becomes true. The whole process is finished after the mechanical claw releases from the airplane and the state goes back to "IDLE".

4.2 Arduino

Chassis Task:



Roboarm Task:



5 Techinical Details

5.1 Communication among Jetson and Arduino boards

We defined our customized protocol as shown in Table 1.

Offset (Byte)	Length (Byte)	Content
0	2	SOF (0x807F)
2	1	Packet ID
3	2	Payload Size
5	N	Payload Data
5+N	2	CRC16
7+N	2	EOF (0x8080)

Table 1: Customized data protocol

The pack and send function is demonstrated in pseudocode in Lists below.

```
// Pack and send data to TX
procedure Send
    buf[0 : 1] = SOF
    buf[2] = id
    buf[3 : 4] = size
    buf[5 : 5+size-1] = data
    crc = crc16(buf[2 : 5+size-1])
    buf[5+size : 6+size] = crc
    buf[7+size : 8+size] = EOF
    send buf to serial
endprocedure

// Receive data from serial buffer
procedure Receive
    append serial buffer to buf
    sof_idx = -1
    eof_idx = -1
    for i = 0 to eff_len(buf) - 2
        if (buf[i:i+1] == SOF)
            sof_idx = i
        else if (buf[i:i+1] == EOF)
            unpack buf[sof_idx+2 : eof_idx-1]
        endif
    endfore
endprocedure

// unpack data into the respective recv_dst
function unpack (data)
    id = data[0]
    size = data[1 : 2]
    calc_crc = crc16(data[0 : 2+size])
    tran_crc = data[3+size : 4+size]
    if (tran_crc == calc_crc)
        recv_dst[id] = data[3 : 2+size]
        if recv_callback[id] != NULL
            recv_callback[id]()
        endif
    endif
endfunction
```

We coded the following function to assign the destination pointer to recv_dst array:

```
int communication_register_recv
(communication_t comm, uint8_t pkt_id, void* data_ptr,
 size_t len, recv_callback_t callback)
```

callback if called if packet of this ID is correctly received. In this way, by using the following code

```
void forward_roboarm_cmd() {
    communication_send(&com_S3, ROBOARM_CMD_ID,
                       &fwd_roboarm_cmd, sizeof(fwd_roboarm_cmd));
}
void forward_sensor_fdbk() {
    communication_send(&com_S2, SENSOR_FDBK_ID,
                       &fwd_sensor_fdbk, sizeof(fwd_sensor_fdbk));
}
void chassis_setup() {
    communication_register_recv(&com_S2, ROBOARM_CMD_ID, &fwd_roboarm_cmd,
            sizeof(fwd_roboarm_cmd), (recv_callback_t)&forward_roboarm_cmd);
    communication_register_recv(&com_S3, SENSOR_FDBK_ID, &fwd_sensor_fdbk,
            sizeof(fwd_sensor_fdbk), (recv_callback_t)&forward_sensor_fdbk);
    ...
```

We can easily realize the data switching between the two (or even more, if needed) Arduino boards.

5.2 Protocol

With the communication module above, data can be easily exchanged by sending and receiving the packed structure. There are 4 types of packets: chassis command, chassis feedback, roboarm command, and sensor feedback.
In Arduino code, they are defined as

5.3 DC Motor speed control with PID

To measure the wheel speed, following macro function is coded to assign the motor and register encoder interrupt.

```
#define MOTOR_ASSIGN(NAME, PIN_1, PIN_2, PIN_PWM, PIN_A, PIN_B)      \
void isr_##NAME();                                                   \
struct motor_device NAME = {                                         \
    .pin_1 = PIN_1, .pin_2 = PIN_2, .pin_pwm = PIN_PWM,              \
    .pin_ecd_A = PIN_A, .pin_ecd_B = PIN_B, .ecd_isr = isr_##NAME    \
};                                                                   \
void isr_##NAME(void) {                                              \
    if(digitalRead(NAME.pin_ecd_B)) NAME.data.total_ecd++;           \
    else NAME.data.total_ecd--;                                      \
}
```

PID library developed by DJI Robomaster is borrowed and applied to control the motors, with the four speed of mecanum as reference and encoder measured speed as feedback. The speed unit is is in RPM, and PID parameters used are

Parameter	Value
k_p	0.8
k_i	0.1
k_d	0
Maximum input error	0
Maximum output	MOTOR_DUTY_MAX (255)
Integral limit	MOTOR_DUTY_MAX / 5 (51)

Table 2: Chassis wheel motor PID params

5.4 Line following

The raw line tracker value is stored in a 8-bit variable as

```
value = (digitalRead(sensor->pin_3) << 3) |
        (digitalRead(sensor->pin_2) << 2) |
        (digitalRead(sensor->pin_1) << 1) |
        (digitalRead(sensor->pin_0));
```

And an error is calculated based this value, if the two outer sensors see black, then use the outer sensor's bias, otherwise, use the two inner sensors.

```
error = (((value & 0b1000) >> 3) - ((value & 0b0001))) * 3;
if (error == 0)
   error = ((value & 0b0100) >> 2) - ((value & 0b0010) >> 1);
```

Use this error as feedback, a PID controller is applied. The reference is always 0, and output is used to get the left and right side wheel speed as

```
left = 1 - pid_out;
right = 1 + pid_out;
```

5.5 Roboarm Calculate

For our 2-DoF roboarm, Joint-1 controls the backward-forward of the arm, which also change the height of the Joint-2. Joint-2 controls the pitch angle of the clamp. Thus the roboarm can be controlled via a height and an angle.
Upon receiving the height, the angle θ of Join-1 servo is computed by

```
h = height - H_OFFSET;
if (h > ARM_LEN)
    h = ARM_LEN;
theta = degrees( acos(h / (ARM_LEN + 0.1)) );
```

5.6 Clamp close

When the "close" flag in roboarm command is 1, the clamp will close. It will first close util the micro switch assert, and then close for a further 20 degrees to make the grabbing tight enough. This is implemented with the help of "clamp_tight" flag. Each loop it will do

```
if switch touched
    clamp_deg = clamp_deg - 1
    servo.set_angle(clamp_deg)
    clamp_tight = false
else if clamp_tight == false
    clamp_deg = clamp_deg - 20
    servo.set_angle(clamp_deg)
    clamp_tight = true
endif
```

6 Reflection

When we are developing our robot, we met several challenges or difficulties. This section will talk about difficulties and our solutions.

6.1 Problem with DetectNet

We mentioned that Computer Vision techniques are adopted to recognize the airplane. However, the process of detecting the airplane is not successful initially due to mainly two factors:

1. Our airplane, especially installed with a handle and a light source, is sometimes not similar to the training dataset of the Detectnet.
2. There may be some interference (i,e., human, chairs) when testing that avoid the model from recognizing the object.

To address the problem, We firstly tried to train our own model with our custom pictures as the training dataset. However, we met several issues such as the version incompatibility of Numpy and Pytorch on Jetson Nano. As a result, although we successfully have the training data, we eventually failed to finish validation and obtain a valid model.
Instead of changing the model, we shift our focus to alter the environment to better fit the original training data from DetectNet. We found that CV actually works well when the airplane is around 1 meter apart from the camera, and thus we will always follow this standard when making the black line and placing the airplane.
In order to remove any probable disturbances, we will also find places with no people appear in the camera. Finally, we observed that it is more easier if the airplane is placed at some angle between the camera. It seems that the confidence rate is expected to increase with the visibility of propeller and the body of the airplane.

6.2 Problem with dragging

After the model locates the airplane, the robot is expected to move towards the robot through light seeker as demonstrated in the required function. The process of dragging after the light seeking is initially difficult to solve because of several challenges:

1. The robots arm's servo motors' torque are too low and it is difficult to control the angle of mechanical claw to the ground. The mechanical claw is initially not able to rose to a certain degree as we wrote in the program. This overload phenomenon may destroy the servo motor and even the servos.
2. The four sensor installed on the mechanical claw are sometime not sensitive enough, but the mechanical claw may fail to hold the airplane tightly if the sensors are too sensitive.

To mitigate the first problem, we added rubber band from robot arm and the Jetson nano board to decrease the load of the servo motor. We also used other things like ties and tape to strengthen the ability of robot arm to hold the heavy objects. In addition, the servo motors are rated to operate at 4.8-6.0V, so we bought a seperated Buck converter to provide around 5.5V voltage to the roboarm servos to make their torque higher than the one with USB powered.
The second problems are relatively harder to solve, because it is time consuming to adjust the distance of various parameters. To achieve our aims, we fix the position of the sensors, make the handle become thicker, and increase the angle of the mechanical claw to better clamp the airplane.

6.3 Problems with Serial communication

The Serial port socket on the Arduino extension board is too lose, and the transmission corrupts from time to time. We applied CRC16 checksum in protocol to eliminate dirty commands which could be dangerous, but timeout from time to time due to bad connections cannot be easily solved.

6.4 Problems with I2C communication with sensors

The bread board never works. The holes are so lose and the I2C communication is interrupted by it from time to time. When request data from I2C bus, if the connection is interrupted, the Arduino will wait forever for the data that should have arrived. As a result the program hangs at the I2C read operation and the car dies.
We identified the problem too late and did not have enough time to make a PCB or solder the Y-lines to share the I2C bus robustly. As a result, in order to at least show you something and avoid a zero-mark in demo, we removed the INA226 and MPU6050 sensors both physically and in program, and connected the BMP and ToF directly to the extension board's two I2C ports to avoid using bread board.