Software Formalization

Year: 2022 Semester: Spring Team: 8 Project: Gimbal vehicle

Creation Date: 2/15/2022 Last Modified: 2/18/2022

Member 1: Zeren Li Email: li3103@purdue.edu

Member 2: Deyuan Sun Email: sun829@purdue.edu

Member 3: Bohang Ni Email: ni67@purdue.edu

Member 4: Haiwen Zhang Email: zhan3237@purdue

Assignment Evaluation:

| **Item** | **Score (0-5)** | **Weight** | **Points** | **Notes** |
| --- | --- | --- | --- | --- |
| **Assignment-Specific Items** | | | | |
| **Third Party Software** |  | x2 |  |  |
| **Description of Components** |  | X3 |  |  |
| **Testing Plan** |  | x3 |  |  |
| **Software Component Diagram** |  | x4 |  |  |
| **Writing-Specific Items** | | | | |
| **Spelling and Grammar** |  | x2 |  |  |
| **Formatting and Citations** |  | x1 |  |  |
| **Figures and Graphs** |  | x2 |  |  |
| **Technical Writing Style** |  | x3 |  |  |
| **Total Score** |  | | |  |

5: Excellent 4: Good 3: Acceptable 2: Poor 1: Very Poor 0: Not attempted

General Comments:

*Relevant overall comments about the paper will be included here*

1.0 Utilization of Third Party Software

We are not planning to use any third party software in our design. Some components in our design would require external libraries to significantly reduce our design time while maximizing our time spent on creativity.

We will utilize a 2.41 Ghz frequency base transmitter, yet we are not going to design the RF circuit or the network algorithm related to it. Or, for communicating with an OLED display, we would like to design the timing and data structure in order for them to communicate, but we are very unlikely to plan to match each English character to the pixel location on the OLED display.

All in all, Our design consists of various major embedded system components and MCU centered modules.

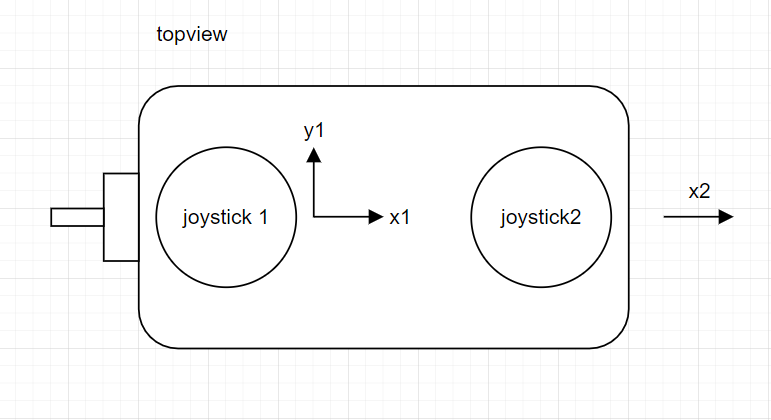
2.0 Description of Software Components

There are many meaningful and significant software components in our design. They are not guaranteed to be optimized, but they are the most feasible and robust solution we can think of so far. Our software components mainly consist of several major parts: controller input, data transmission, wheel control, gimbal control, and Mecanum kinetics. Many explanations will be done using descriptive pseudo code.

**2.1 controller input**

Key Points for this module:

* Joystick is consist of a potentiometer with two axes.
* We convert potentiometer voltage readings into a data structure using ADC.
* Small tilt angle on joystick would be ignored
* we change the mode of the vehicle using a switch on the side.



Each data structure to be sent through the transmitter would contain a value of x1, y1, and x2.

typedef struct controller\_package\_to\_be\_sent{

int x1;

int y1;

int x2;

}pack

we will then send this controller\_package using a 2.41Ghz transmitter using SPI protocol on both transmitter and receiver.

We know that the input potentiometer reading value joysticks using ADC is 12 bits, ranging from 0 to 4095. However, we need a “activation value” on the joystick for it not to be so sensitive. So we filter out the middle 1000 values and treat them as “0”. But we will not directly assign a 0 to them, because 0-4095 is actually mapping to the positive and negative range of moving speed. Instead, each time we read a value that is in between (4095/2) plus or minus 500, we will treat them as standing still.

while{

//otherthings

pack A = received\_from\_controller() // using SPI

A = true\_move(A) // if value between(1547, 2547): value = 4095/2

}

We implement this in a similar but more concrete way, this is for demonstration only.

**2.2 wheel control**

Key Points for this module:

* Our chassis consists of a body and four wheels.
* We cannot change the orientation(not rotation) of each wheel.
* We can change the speed of each wheel.

Each wheel is driven by an individual motor. However, we are not writing codes to control the motor directly. Instead, we write code to generate PWM waves that send to a motor controller. The controller then urges the wheels at the RPM we wish.

NOTE: Input PWM duty cycle valid value from 50% to 100%, 50% will make the wheel start to rotate, and 100% will make the wheel rotate at maximum speed.

**2.3 Mecanum kinetics**

Key points for this module:

* Achieve omnidirectional move.
* Again, we cannot change the orientation of each wheel.
* But, we can change the speed of each wheel.

Each real input value will be transferred into the PWM wave from the 50% to 100% range of all four wheels. However, many of the time our input signifies more things than just moving in x or y direction. It could move slanted, with a curve, or with both. (That’s why it’s called omnidirectional move), so we really would need an algorithm and perhaps some data structures to implement a mapping between the controller input space into wheels’ speed output space.

We receive data x1,y1,x2 (x,y correspond to the horizontal and vertical location of a joystick. 1 is the right joystick and 2 is the left joystick) when we push the right and left joysticks. We design an algorithm to decompose the digital data into four different integers. Each integer will be sent to a motor by PWM to control the speed of its designated Mecanum wheel. [2]

left\_front\_wheel = x1 + x2 + y1

right\_front\_wheel = x1 - x2 - y1

left\_back\_wheel = x1 + x2 - y1

right\_back\_wheel = x1 - x2 + y1

each wheel then rescale to ⅓ to map the PWM duty cycle range (below 100%)

**2.4 Gimbal Control**

Key points for this module:

* Remain still to chassis
* Remain stable to the ground level (self-balance)
* Simple control mode

1. Lock both servos, and we will make the gimbal stay still to the chassis.
2. self-balance mode is more complicated.
   1. First, we need to read an IMU data using I2C, each of the x, y, z values would correspond to the pitch, roll, yaw motion respectively.
   2. We need to map x, y, z values of the IMU back to the input of the servos.
      1. We used 2 axis servos, so we only feedback the pitch and yaw angles.
   3. PWM input to servos:
      1. servo is not like a motor, we can directly input PWM duty cycle to it to control the angle it points to.
3. simple control mode:
   1. we will need to map the input from controller\_package(0-4095 adc reading values) to the current angle position of the servos.

NOTE: our servos have orientation degrees ranging from 0 - 180 only, which is positive 90 to negative 90 degrees. A software based low-pass filter is implemented to reduce glitchy moves on the servos and gimbal.

3.0 Testing Plan

*For the software components detailed in section 2.0, provide further description on a component-by-component basis about what tests need to be performed on the component. Additionally, assign a numerical priority for each component concerning the importance of successfully verifying that component.*

**3.1 Controller input:** The first component to be tested will be the Controller inputs(Joystick ADC reading). This could be tested through the debugger feature of the integrated development environment (IDE).

1. Begin the debugger ,switch to the live expression panel and read the ADC value.
2. Send the ADC value to the Wireless transceiver module, for each sending, blink the LED once.
3. Reading the fifo status register of the Wireless transceiver module[1], make sure it is empty after each sending.

These tests should make sure that all the Controller features are working appropriately.

**3.2 wheel control:** The second component to be tested will be the wheel control.

1. Generate PWM waves with different duty cycles, and observe the speed of the wheel.

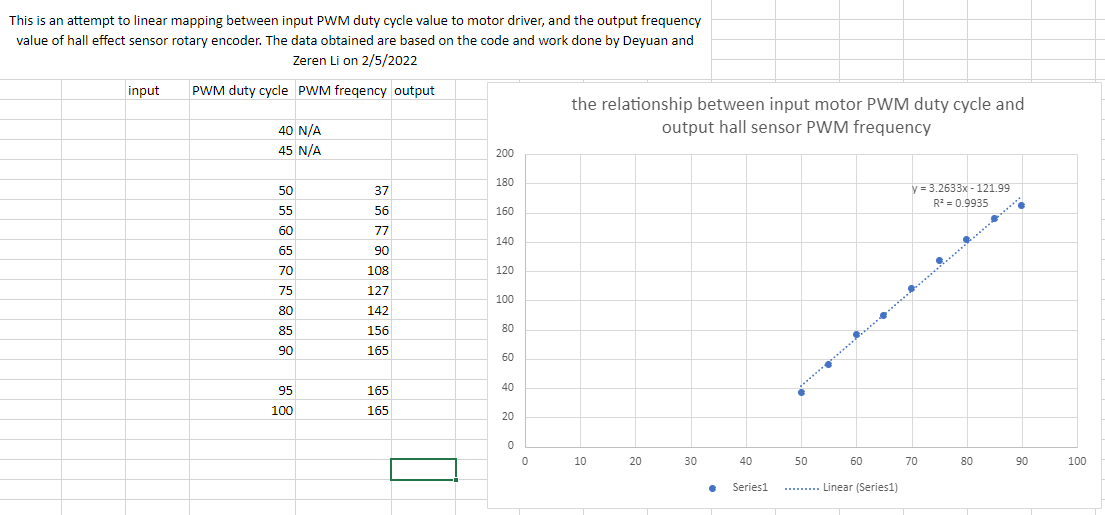
If the wheel is running at the maximum speed, the duty cycle that PWM sends to the wheel will be 90-100 %. The RPM of the wheel will be 330. In the stationary state, the duty cycle is below 50 %. The RPM of the wheel will be 0. The RPM and duty cycle has a relation RPM = duty cycle \* 6.5267 - 243.98.

1. Latency test is about the response time between pushing a joystick and the time when wheels start to rotate. The test includes human judgement. If the person doesn’t feel much latency after he/she pushed the joystick, we will consider the vehicle passed the test.

**3.3 Mecanum kinetics:**

We tried to map a PWM\_in duty cycle input space into an output of hall effect sensor PWM frequency. After two weeks of testing, we eventually got an efficient way to measure the real time hall sensor data.

As information given, we knew that the minimal wheel RPM is 0 and maximum is 330.



Supporting diagram for kinetics

Our result of PWM frequency is the inner part RPS. After looking at the datasheet, we realized that there is a reduction ratio of 30. That is, our value of maximum 165 should be multiplied by 60 and divided by 30 converting to RPM, which is equivalent to 330 RPM.

The equation at right of the figure also provided the validity of our method, so we believed that we acquired a strong method to get the real time RPM.

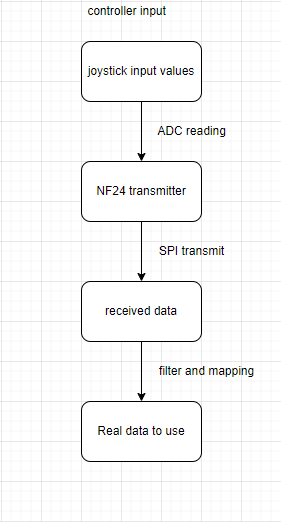
**3.4 Gimbal Control**: The last software component that will be tested is the gimbal control software. The gimbal control software has two main parts. Each part can be tested separately or together. The first main part is the function that accepts the input of a value and adjusts the servo position based on the value. All the values inputted to the function “adjust\_servo\_position\_z” and “adjust\_servo\_position\_x” will be mapped to the range of 25-125 (frequency of PWM). 25 corresponds to 0 degree of the gimbal and 125 corresponds to 180 degrees. In order to test these two functions, we can input multiple values between 25 and 125 to these functions and use tools to physically measure the angle of the servo position to test the correctness of these two functions. The other main part of the gimbal software is in charge of reading data from the IMU. The IMU ought to return the gyroscope data and accelerometer data, based on these data, we can calculate the Yaw, Roll, and Pitch angle of the IMU. To test the correctness of these readings and calculations, we can also use a tool to physically measure the angles of IMU and compare them with the calculated angles. In order to visualize the real-time calculated angles, we incorporated an OLED display to display the angles data. Another thing we need to test is the communication between the two parts. We need to make sure the calculated angles are successfully and correctly input to the function for adjusting servo position. To test this, we can measure both the angle of the IMU and the angle of the servo to see if they are the same. If they are the same, then everything seems correct.

4.0 Sources Cited:

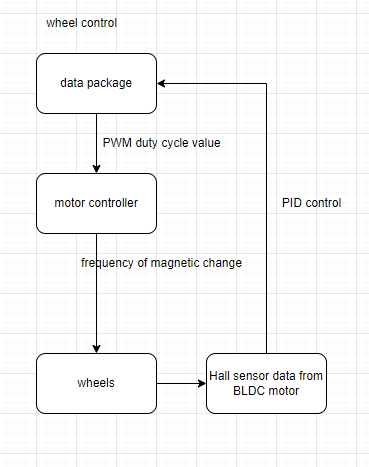
sparkfun. (n.d.). *Preliminary product specification V1 - sparkfun electronics*. datasheets of nRF24L01. Retrieved February 19, 2022, from https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss\_Preliminary\_Product\_Specification\_v1\_0.pdf

Google. (n.d.). *US20140232174A1 - Mecanum wheel and Mecanum-wheeled vehicle*. Google Patents. Retrieved February 19, 2022, from https://patents.google.com/patent/US20140232174A1/en

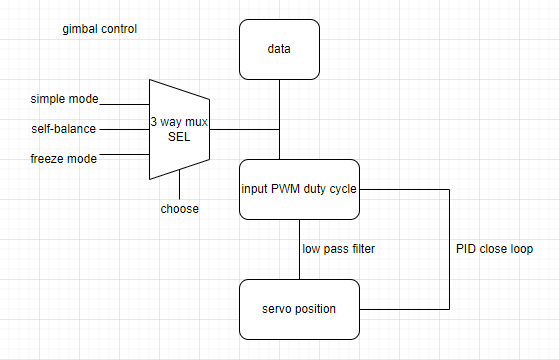
Appendix 1: Software Component Diagram



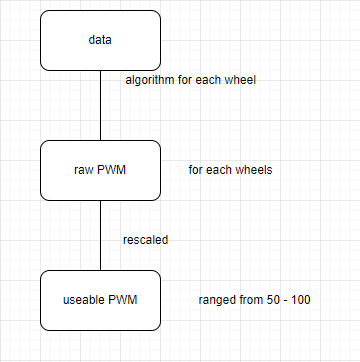
*Figure 1, flow chart of controller input part*

**

*Figure 2, flow chart of wheel control*

**

*Figure 3, flow chart of gimbal control*

**

*Figure 4, flow chart of Mecanum kinetics*