Technical Summary of
Raspberry Pi Tank with Camera + Automatic Fire + Turret

By: Yifan Liu

Executive Summary

I. Problem

The project goal aimed to develop a Raspberry Pi-powered tank with the capabilities of driving (forward, backward, turning left, and turning right), firing a Nerf gun with the push of a key, turning a turntable/turret carrying the Nerf gun, elevating the Nerf gun, and implementing a working camera. Essentially, the project was initially conceived as a hardware-heavy robotics project. Due to the lack of a 3D printer, no custom components could be implemented. The components of the project had to be attached to the Raspberry Pi and work in an efficient manner. Software needed to be written to support the functionality of each component.

II. Solution

The solution for this project consisted of numerous successes and failures, and the building of the project took place in three main steps, with many testing phases within these main steps. The first main step consisted of acquiring the appropriate components for the project. The following is a list of all the components of the project.

- 2x servos (One for turret turning and the other for firing of the Nerf gun).
- 1x chassis (Includes the aluminum frame, wheels, and tracks)
- 2x motors 9v (These were included in the chassis purchase)
- 1x Nerf Jolt (A lightweight, simple Nerf gun)
- Gorilla Tape (Holds together wires, Nerf gun, and servo)
- 1x acrylic plate (Serves as the turntable/turret and attaches to the turret turning servo)
- 1x Pi Camera (Allows for the stream and ML programs to be possible)
- 1x Raspberry Pi (Central command computer for all components)
- 6x 9v batteries (These powered the motor controller and motors)
- 6x 9v battery clips (These allowed for the batteries to be connected in parallel)
- 1x L298N (Motor controller module)
- Male-to-female jumper cables (Connects all components)
- Male-to-Male jumper cables (Soldered to the battery clips and connected to L298N)
- Raspberry Pi battery pack (Provides power for the Pi when the tank is operational)

The second main step was related to attaching the components to the chassis, which were connected using the jumper cables. The optimal amounts for each component were determined and executed efficiently.

The final stage of the project consisted of writing software or utilizing existing software that supported the command of the Raspberry Pi tank wirelessly from a PC or laptop that was connected through VNC viewer to the Raspberry Pi.

Top Five Requirements and Technical Solutions

I. Directional Control

This requirement focused on making the Raspberry Pi tank capable of driving forward, reversing, turning left, and turning right. Accomplishing this step consisted of driving 4 holes to mount the Raspberry Pi to the underside of the chassis, soldering 6 battery clips together in parallel, and connecting the Pi, motors, and batteries using a motor controller. Code was then written that could translate string values into directional movement. Motor configurations were initialized and put into if/else statements.

II. Turret Turning

The turret turning requirement designated that the gun could fire at different pre-specified angles. To accomplish this requirement, 2 holes were drilled into the chassis, allowing for the mounting of a servo to the underside of the chassis. The servo would connect to a mounting piece attached to the bottom of the acrylic plate. After these components were attached, it was determined that the servo could turn weights up to 1 lb. This testing confirmed that it could carry the weight of a servo and Nerf gun and still turn. Code was written that turned the turret to preset angles, i.e., 5, 30, 50, and 100 degrees.

Sub Requirement: The tank can simultaneously drive and turn the turret.

III. Auto Nerf Gun Firing

The third requirement revolved around automating the firing process of the Nerf gun mounted on the acrylic plate. Using tape, a second servo and a Nerf gun were mounted to the acrylic plate. The jumper cables connected to this servo were doubled in length to account for the turning of the turret. Code was written that turned the servo from 40 degrees to 140 degrees and back to 40 degrees in 1 second. This process helped reduce the tension that the Nerf gun and servo suffered.

Sub Requirement: The tank can simultaneously drive and fire the gun.
Both sub-requirements were accomplished through the use of reverse string indexing.

IV. Flask Server for Camera

The fourth requirement for the Raspberry Pi tank was to implement a live stream for the Pi camera module. This requirement was accomplished by downloading numerous dependency files through Linux commands and cloning an existing Git Repository linked below.
https://github.com/EbenKouao/pi-camera-stream-flask

V. Object Classification for Camera

The fifth requirement for the Raspberry Pi tank was the addition of an object detection model for the Pi camera module. It was determined that it would be far more time efficient to use a pre-trained object classification model. By downloading numerous packages such as TensorFlow Lite and Utils, the existing cloned repository of the project was fully functional.
https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/raspberry_pi

Failures + Challenges

1.) One notable failure was related to an occurrence on project filming day. One of the Pi's GPIO ports, specifically BCM GPIO 4, proceeded to fail to send signals. Being unable to initially diagnose this error, the issue quickly became a formidable threat to derailing the entire directional control requirement of the project. With this error, one orientation of high and low GPIO signals for one motor failed to function properly. As a result, when the forward command was executed only one side of the tracks would spin. This situation was also the case for turning right. Meanwhile, full functionality was still present for the reverse and turn left directional controls since these commands utilized the other orientation of high and low GPIO signals. The resolution process for this issue began with the removal of the problematic track and putting it back on. Unfortunately, this fix did nothing to resolve the issue. At this point, the issue was believed to originate from a faulty or recently damaged motor controller. The motor controller was thus replaced with a backup. Afterward, it was apparent that the problem persisted. Upon using a voltmeter to test the voltage that the motor controller emitted, it was determined that a signal of 0 volts was emitted during the problematic orientation. Therefore, it was likely that a GPIO port could have failed. Each wire connecting a GPIO pin to the motor controller was moved one by one to different ports and simultaneously tested to determine which port was the problem. By moving the wire connected to the GPIO 4 pin, the issue was finally resolved. This form of resolving challenges helped me improve my problem-solving capabilities in dealing with electronics and hardware. Despite the arduous process of determining the error, this fixing process was ultimately a success.

2.) Another major failure/challenge that occurred early in the planning stages of the project was finding a method to best implement the elevation mechanism. This feature was eventually scrapped due to the complexities that it introduced and the lack of a third servo. Originally the plan was to attach a stick to a servo mounted flat on the turret. When the servo turned upward, the gun would be elevated. To substitute this requirement, I began to research the best ways to implement object classification and a live stream

capability for the Raspberry Pi tank. The key takeaway in this situation was to always have a backup plan. Despite the functional status of the object classifier, there lacked time to implement a notable feature, where if the bounding box label was set to a person, then the Nerf gun would autofire. If I had the time, my solution to this problem would have been to combine the detect.py program with my main project program. Then, using a function or variable to store the value of the label, an if statement could hold the code to fire the gun.

3.) Another major failure/challenge was the attachment of the turret turning servo and the Raspberry Pi onto the chassis. Due to having never operated a drill before, this phase of the project proved quite difficult. Essentially, the chassis lacked holes that could allow for a different Raspberry Pi orientation. Moreover, the servo needed to be securely attached to the underside of the chassis to support the weight of the turret. Several near accidents occurred as I was familiarizing myself with using the drill. A notable error that occurred during the drilling was that the location of one of the 4 holes meant to secure the Pi in place was drilled in the wrong place. This issue resulted in the Pi having to be held in place with the remaining 3 correctly drilled holes. Another failure was that the nuts used to attach one side of the servo interfered with the rotation of the turret. As a result, these nuts and bolts were removed, and one side of the servo was held together with tape instead. One major takeaway and accomplishment gained from this experience is the capability to confidently operate a drill. Another triumph in overcoming this challenge was that the improvised taped side of the servo worked nearly as well as with the bolts.

Successes

1.) To succeed with the implementation of the trigger mechanism of the servo fire capability of the Nerf gun, a continuous trial and error testing method was utilized. The feasibility of the firing mechanism was first tested off the tank and an initial build for the turret was created. However, this version suffered from weight imbalance, and since the barrel of the Nerf gun extended out of the turntable, turret turning was affected when the gun was oriented over the tracks. The second model was the final variant, where the Nerf gun was

shifted on the turntable. The servo in this variant was attached at a slanted angle on the turret to eliminate issues during turret turning. This new servo placement also improved the weight balance issue. Ultimately, extensive testing was required to determine the tightness of the wire around the Nerf gun trigger. Since the servo and Nerf gun were held in place with the help of duck tape, tension needed to be also not too high for a lengthy period. Substantial tension risked pulling the servo up from the tapped down position. Eventually, after experimenting with one wire loop around the trigger, it was determined that multiple loops would improve stability while firing. Three loops with a solid knot provided the most grip on the Nerf gun. As a result, the Nerf gun could be successfully fired every time the command was given. Moreover, the current implementation worked well with the servo that turned the turret. This part of the project helped me learn how to control servos.

2.) The implementation of the string indexing system proved to be quite successful. In essence, both sub-requirements for II and III utilize negative indexing of the last element of the string to simultaneously execute 2 commands. Consequently, it was possible to type in commands like wf while the previous w (forward command) was in execution. This system allowed for the tank to be able to turn its turret while driving and also fire while driving. Both of these sub-requirements could also serve as main requirements. The use of this system provided the opportunity for the most simple implementation of these requirements. Considering the numerous hardware challenges encountered, it was essential keeping the code as simple as possible. The takeaway here is that a loop with some conditional statements proves efficient in the face of complexity.

3.) Another major success was the effective use of battery clips in place of a battery pack. This implementation helped cut the costs of the project. However, a negative of this implementation was that soldering the battery clip wires in parallel became essential to the success of providing stable motor power. A positive of this implementation was that I learned how to effectively solder wires together. Another advantage of using battery clips was that I could easily add or remove batteries to the circuit depending on the current needed by the motors.

```
#(@author): Yifan Liu
#PSU Email: ypl5702@psu.edu
#Assignment name: Final Project
#Created: Tuesday November 15, 09:55:17 2022

import RPi.GPIO as GPIO
from time import sleep
import time
import curses

#in1 = 24
#in2 = 23
#en = 25
#change

in2 = 6
in1 = 5
en = 25

in3 = 3
#in4 = 4
enb = 2

in4 = 26

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

#left side
GPIO.setup(in1,GPIO.OUT)
GPIO.setup(in2,GPIO.OUT)
```

```
GPIO.setup(en,GPIO.OUT)
pwm = GPIO.PWM(en, 1000)

#GPIO.setup(in1,GPIO.OUT)
#GPIO.setup(in2,GPIO.OUT)
#GPIO.setup(enb,GPIO.OUT)
#pwm = GPIO.PWM(enb, 1000)

GPIO.setup(in3,GPIO.OUT)
GPIO.setup(in4,GPIO.OUT)
GPIO.setup(enb,GPIO.OUT)
pwm1 = GPIO.PWM(enb,1000)

# GPIO.setup(in3,GPIO.OUT)
# GPIO.setup(in4,GPIO.OUT)
# GPIO.setup(en ,GPIO.OUT)
# pwm1 = GPIO.PWM(en,1000)

s1 = 17
GPIO.setmode(GPIO.BCM)

#GPIO.setwarnings(False)

GPIO.setup(s1,GPIO.OUT)
servo = GPIO.PWM(s1, 50)
servo.start(0)



s2 = 27
```

```
GPIO.setup(s2,GPIO.OUT)
servo1 = GPIO.PWM(s2, 50)
servo1.start(0)

pwm.start(0)
pwm1.start(0)



while(True):
    var = input("Enter direction: ")
    if len(var)>1:
        if var[-1]=='j':
            servo.ChangeDutyCycle(2+(5/18))
            time.sleep(0.5)
            servo.ChangeDutyCycle(0)
        elif var[-1]=='k':
            servo.ChangeDutyCycle(2+(30/18))
            time.sleep(0.5)
            servo.ChangeDutyCycle(0)
        elif var[-1]=='l':
            servo.ChangeDutyCycle(2+(50/18))
            time.sleep(0.5)
            servo.ChangeDutyCycle(0)
        elif var[-1]=='m':
            servo.ChangeDutyCycle(2+(100/18))
            time.sleep(0.5)
            servo.ChangeDutyCycle(0)
        elif var[-1]=='f':
            servo1.ChangeDutyCycle(2+(40/18))
            time.sleep(0.5)
            servo1.ChangeDutyCycle(2+(140/18))
```

```
        time.sleep(0.5)
        servo1.ChangeDutyCycle(2+(40/18))
        time.sleep(0.5)
        servo1.ChangeDutyCycle(0)
    var = var[0]
#forward
if var == 'w':
    #right
    GPIO.output(in1, GPIO.HIGH)
    GPIO.output(in2, GPIO.LOW)
    pwm.ChangeDutyCycle(100)

    #left
    GPIO.output(in3, GPIO.LOW)
    GPIO.output(in4, GPIO.HIGH)
    pwm1.ChangeDutyCycle(100)
#stop
elif var == 'q':
    #right
    GPIO.output(in1, GPIO.HIGH)
    GPIO.output(in2, GPIO.LOW)
    pwm.ChangeDutyCycle(0)

    #left
    GPIO.output(in3, GPIO.LOW)
    GPIO.output(in4, GPIO.HIGH)
    pwm1.ChangeDutyCycle(0)
#backup
elif var == 's':
    #right
    GPIO.output(in1, GPIO.LOW)
```

```
    GPIO.output(in2, GPIO.HIGH)
    pwm.ChangeDutyCycle(100)

    #left
    GPIO.output(in3, GPIO.HIGH)
    GPIO.output(in4, GPIO.LOW)
    pwm1.ChangeDutyCycle(100)
#turn right
elif var == 'd':
    #right
    GPIO.output(in1, GPIO.LOW)
    GPIO.output(in2, GPIO.HIGH)
    pwm.ChangeDutyCycle(100)

    #left
    GPIO.output(in3, GPIO.LOW)
    GPIO.output(in4, GPIO.HIGH)
    pwm1.ChangeDutyCycle(100)
#turn left
elif var == 'a':
    #right
    GPIO.output(in1, GPIO.HIGH)
    GPIO.output(in2, GPIO.LOW)
    pwm.ChangeDutyCycle(100)

    #left
    GPIO.output(in3, GPIO.HIGH)
    GPIO.output(in4, GPIO.LOW)
    pwm1.ChangeDutyCycle(100)
```