Final Project Report
EC-535: Introduction to Embedded Systems
Boston University
Team Follow Me

Alejandro Aparicio (aaparici)
Muhammad Kasim Patel (kasimp93)
Shangqiu Cai (leocsq)

Abstract

The Follow Me project is a prototype embedded system that would help people achieve higher mobility near or around their home by providing a small electric car that will help carry items for you and follow wherever you go. With the use of a Gumstix Verdex, and Raspberry Pi and the software development done in C for Linux Kernel, C++ and Python we were able to build a prototype car that would follow a target. The Follow Me project is a prototype embedded system that would help people achieve higher mobility near or around their home by providing a small electric car that will help carry items for you and follow wherever you go. With the use of a Gumstix Verdex, and Raspberry Pi and the software development done in C for Linux Kernel, C++ and Python we were able to build a prototype car that would follow a target.

1. Introduction

This project was developed to create a useful embedded system product much like the products already currently developed by putting in practice the recent skills developed during our Introduction to Embedded Systems Class. The embedded system developed is a prototype implementation of products already in the market that will identify a person or a target like a keychain and will follow the user wherever they go. Examples of this are a suitcase that will identify a key chain or "leash" device and follow it around so the user is free to move without carrying or dragging the suitcase[1]. Another example is a Cambridge Massachusetts startup that developed a cart that will follow the user around and can even do some shopping for you[2] and we identified ourselves more with this prototype which can be leveraged to different concepts.

This project with the right safeguard can be easily applied to other items like strollers, personal shopping carts or even bicycle carts to help push, carry or drag whatever necessary.

Depending on the application it can help elderly people or people with physical disabilities become more independent. For other more convenient applications it can help busy parents or any individual to carry groceries around from the car or a nearby grocery shop.

This project is important since it also helps the students to think about integration of various systems like a vision component, scheduling and integration and communication of different developing languages as well as hardware constraints.

Before we started to code we needed to specify the hardware and limitations we might encounter in the development of the project. One of the main guidelines was to achieve the main objective of following a target with a very limited time. With this in mind we decided to use imaging

processing based on one of our teammates developing experience in vision and the other teammates effort in communications, testing and Linux Kernel Driver Development acquired during class.

The team successfully build a car that can move independently to follow a specific target within a specific range. One of the objectives was also speed which we believe was achieved successfully by correctly scheduling interactions between the vision software in the Raspberry Pi, a task scheduler to retrieve the information from the vision program and send it to the Gumstix for hardware control. We were able to improve speed functionality from 1 second response to 0.5 seconds response by applying a task scheduler and communication functionalities. We also were able to improve performance and reduce software crashing by applying fault accommodation in the data interpretation and transmission software.

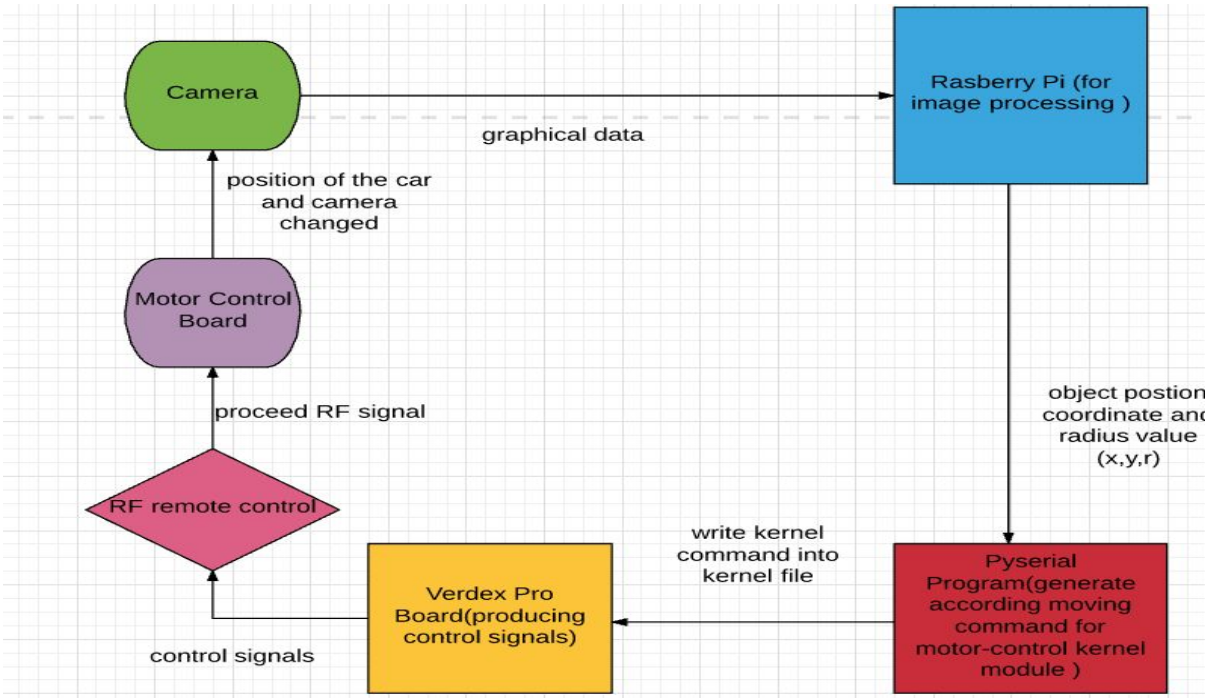

Fig.1 The above figure shows the flow diagram of all the modules used in the project

2. Design Overview

The prototype hardware is composed of two DC motors, one RF communication controller, one Raspberry Pi, One Gumstix Verdex and one Camera. For Software we used one Kernel Device Driver programmed in C to control the motors and direction a C++ program with grayscale and noise reduction functions and for communication from Raspberry to Gumstix we used Python pyserial library.

As part of the original design we also decided to use a iRobot Create as the motor drives and have a stretch objective to use the additional infrared sensors to detects object and stairs. However a

major issue we had with the Gumstix was to develop asynchronous serial driver communication. After brainstorming different option like controlling the iRobot's motors directly, we decided to move on to a different type of hardware and get a RF car where we can control the motors directly by the GPIO's of the gumstix based on previous Labs work.

Fig.2 Follow Me prototype hardware composed of a Gumstix, Raspberry Pi, Car and Power Sources.

Once the hardware was set, additional design considerations needed to be made like how the target recognition program should recognize the position and how the information would be transferred and processed to the gumstix.

Since the motor for the car and the gumstix available power output from the GPIO is different we needed to implement an H bridge, or an optoisolator triac driver which we only had 1 available and little time to order and implement. As a solution we decided to use RF communication which will provide the correct power isolation for the gumstix and the right control to the cars motors. The reason we decided to get this route was due to ease of implementation and time available to implement.

For the task implementations we divided the tasks in two people, so one can be the main person of contact and an additional person to help where necessary. Please refer to the following table to see tasks and time assignments:

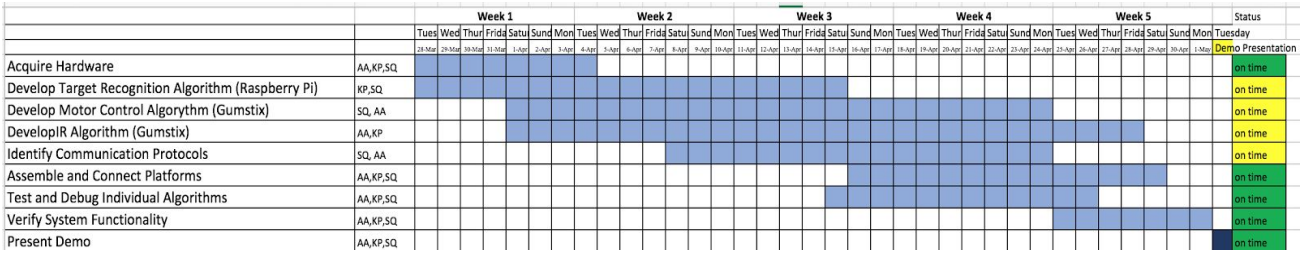

Table 1 shows the division of tasks among the team members where AA is Alejandro Aparicio, KP is Muhammad Kasim Patel, and SQ is Leo.

The previous chart accurately shows responsibilities assigned and developed, however the tasks 2 to 5 were delayed due to communication issues with the iRobot.

- Kasim developed the Vision aspect and C++ object recognition program.
- Kasim setup the Raspberry Pi and the Raspberry Pi Camera Hardware.
- Leo developed the Linux Kernel Driver
- Alejandro developed the serial communication driver.
- Leo and Alejandro design the hardware implementation.
- Alejandro Aparicio designed vision program object interpretation X, Y and Z, testing and fault accommodation.
- All three were involved in prototype testing and tweaking software and hardware for a better synergy.

3. Project Details

(a) **Object Detection** (See /Object_Detection/Source.cpp for source code)

We implemented object detection using OpenCV computer vision libraries [3]. We set the resolution of the camera to be 640x380. The reason for setting this resolution was to get a wider field of view for the camera. As a tradeoff between processing power of raspberry and to get the optimum viewing angle for our car we set the resolution.

The Hough Circle Transform works in a roughly analogous way to the Hough Line Transform. We need three parameters to define a circle:

$$C : (x_{center}, y_{center}, r)$$

where (x_{center}, y_{center}) define the center position (gree point) and r is the radius, which allows us to completely define a circle as seen below
Our Algorithm:

- Loads an image
- Blurs it to do noise reduction in order to reduce false circle detection
- converts the image to grayscale

- Applies the Hough Circle Transform to the blurred image .
- Displays the detected circle in a window
- The coordinates of the center of the circle and the radius are written into a file which is then read by the gumstix.

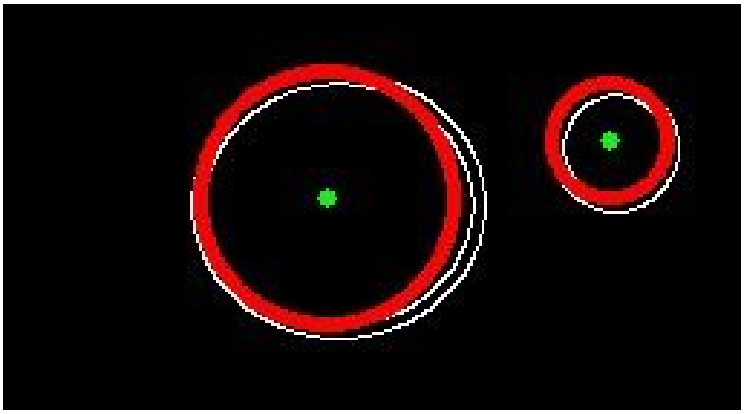

Fig. 3 The red circle shows the boundary of circle detected by our program and the green dot shows the center of the circle as detected by our Object Detection algorithm.

(b) Gumstix kernel module (See /km/fmkm.c and /km/Makefile for the source code)

The main task for the kernel module is receiving command from pyserial and generating RF remote control signals to remote controller by GPIO ports.

Basically, for the car moving, the commands should be able to splitted into five kinds: straight forward, straight backward, turn right, turn left and stop. More specifically, for commands that consist of both moving and turning, instead of sliding left or right directly, we need to support forward left, forward right, backward left and backward right. In terms of command realization in kernel module, we use the read and write ops of the kernel to insert and identify command.

The format of the command we get from pyserial is simple: there are three kinds of opcode: “m0,1,2” represents stop, forwarding and backwarding; “d0,1,2,3,4” represents stop, forward left, forward right, backward left and backward right ; “v0,1,2” represents high, medium and low motor speed. When the kernel module file gets new command from pyserial, the kernel will call write function to translate the command into values for variables ctrl[4] which represent operating status of the motor; In the meantime, the read function will print the command to the screen for monitoring.

For manipulating the operation time for each command, we use a periodically updated timer. When the timer expires, the timer will check the values of each variables ctrl that represent operating status of the motor, and then proceed according PWM and GPIO signals for remote control module. About specific expire time for the timer, after multiple times of trying , we finally

set it as 0.2s, which works well. We have tried the period in range of 0.1 to 0.75s, and 0.2s can both keep the car moving continually and prevent it from getting confusion because of high frequency of commands.

After accomplishing the task of basic movement, we also add one more command "f0" for indicating out-of-range fault. We plan to set a led indicator light , while receiving the "f0" command, the kernel will proceed "stop" signal first and then turn on the indicator light so that the holder will know that the car can't find its target in its range.

(c) Serial Communication (See /SCFD/gumstix_comm.py for the source code)

An important part of the project is the ability and for the Raspberry Pi Visual detection program to communicate to the Gumstix the X and Y position of the target as well as the size of the radius of the circle which depending on the size will tell us if the car needs to move forward or backward.

After some consideration we decided that the C++ program will write the coordinates to a text file in the Raspberry Pi and a python program will translate the coordinates to commands to the gumstix so in turn controls the motors. The main reason for using Python that it provided with an easy implementation of the asynchronous serial communication[4] and the commands could be directly sent to the /dev/fmkm kernel module.

Scheduling transmission was key to allow the C++ program to detect and write the coordinates as well as to send it to the gumstix with some delay to compensate for motor dynamic, hysteresis and reaction speed. Without a doubt the decision to write to a csv and error detection became the bottleneck of the timing scheduling. Out min time of reaction was 0.3 seconds however anything below that the programs will become unstable.

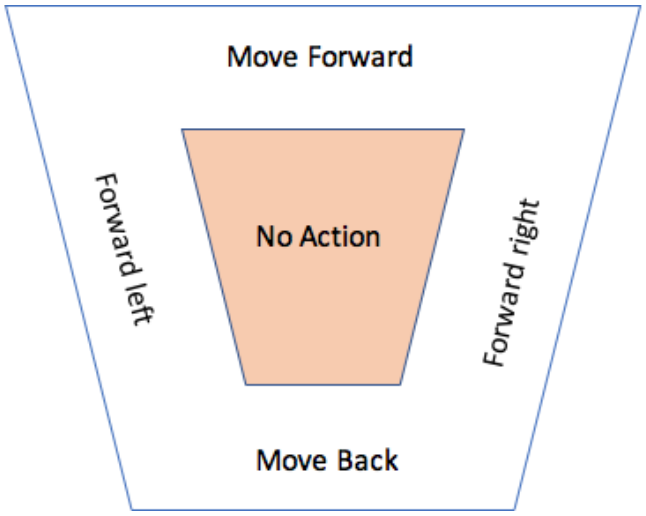

Fig. 4 Field of vision and reaction interpretation done in the python program.

(d) Error Detection and Testing

A major part of the project was to test and use programming techniques for fault detection, fault announcement and fault accommodation.

Range Test. With the Object detection program know reliability we decided to exercise the full range of visual field from the camera, and we even added point outside the range to see how the gumstix will react. The solution was to filter any non number commands.

False Positive Test. Once the target moves out of range the object detection program will start detecting False Positives, however they will not always in the same spot unless it detects another circle. The solution for this is to do an overrate detection with respect to the radius of the circle and the x and y change.

Fault 1: If abs (current radius - previous radius) > 50 pixel send fault

Fault 2: If abs (x range radius - previous x) > 30 pixels send fault

Fault 3: If radius last seen in a radius greater than 215 pixels, move back

Additional Fault accommodation we would like to implement but it would slow our reaction even more was to get a 3 sigma position over a period of 1 second, that way we can almost certainly guarantee the command sent is accurate.

(e) Programming Languages Used

- OpenCV C++ for Object Detection
- C++ for gumstix kernel module
- Python for communication from Raspberry to Gumstix

(f) Performance of each Software component

Performance	Time(s)
Object Detect Delay (varies depending on lighting conditions)	0.2
Pyserial Command Sleeptime	0.4
Kernel Timer period	0.2
Command Delay In Total	0.5

Table 2 shows the performance of each individual software component

4. Summary

Follow Me car is a prototype implementation of a tool that could help people carry things around when shopping, running or in the move. We implemented an electric car which follows a target and moves depending on the target distance and position.

Summarize your project topic and your implementation. What are your accomplishments? What are some of the remaining challenges about this project?

References

[1] McFarland, Matt, This Suitcase Will Follow You Around Like a Puppy, CNN, *http://money.cnn.com/2016/10/18/technology/suitcase-autonomous-follow/*

[2] Pierce, David, The Cute Robot That Follow You Around And Schleps All Your Stuff, *https://www.wired.com/2017/02/piaggio-gita-drone/*

[3] OpenCV Hough Circle Transform *http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html*

[4] Opening Serial Ports, *http://pyserial.readthedocs.io/en/latest/shortintro.html*