ECE-544: Final Project (Automated Robotic Can Detection)

TEAM MEMBERS:-

Samuel Burkhart:- Hardware Design(Motor and Servo Arm), C\Python level drivers

Dakota Ward:- Can Detection using Haar Cascade Classifier, Distance Measurement using Triangle Similarity(Camera Calibration).

Khyati Sinha:- Color Detection using OpenCV and Bounding Box(contours) around the detected object.

Introduction

The Automated Robotic Can Retrieval is a robot control designed on Digilent's new product, the Pynq-Z1 board. The robot utilizes computer vision techniques for the can detection ,distance measurement and for the color detection .

The robot utilized the Anaconda distribution of Python and the OpenCV packages for implementation of Computer Vision in our Design.

Goal: Design a robot to identify a can of a given color and take it to a target location.

Peripherals Used

Digilent/Xilinx Pynq Board: Zynq + ARM design using a mixture of HDL/C/Python to enable unique embedded system designs

Actobotics Runt Rover Junior: Cheap robot platform

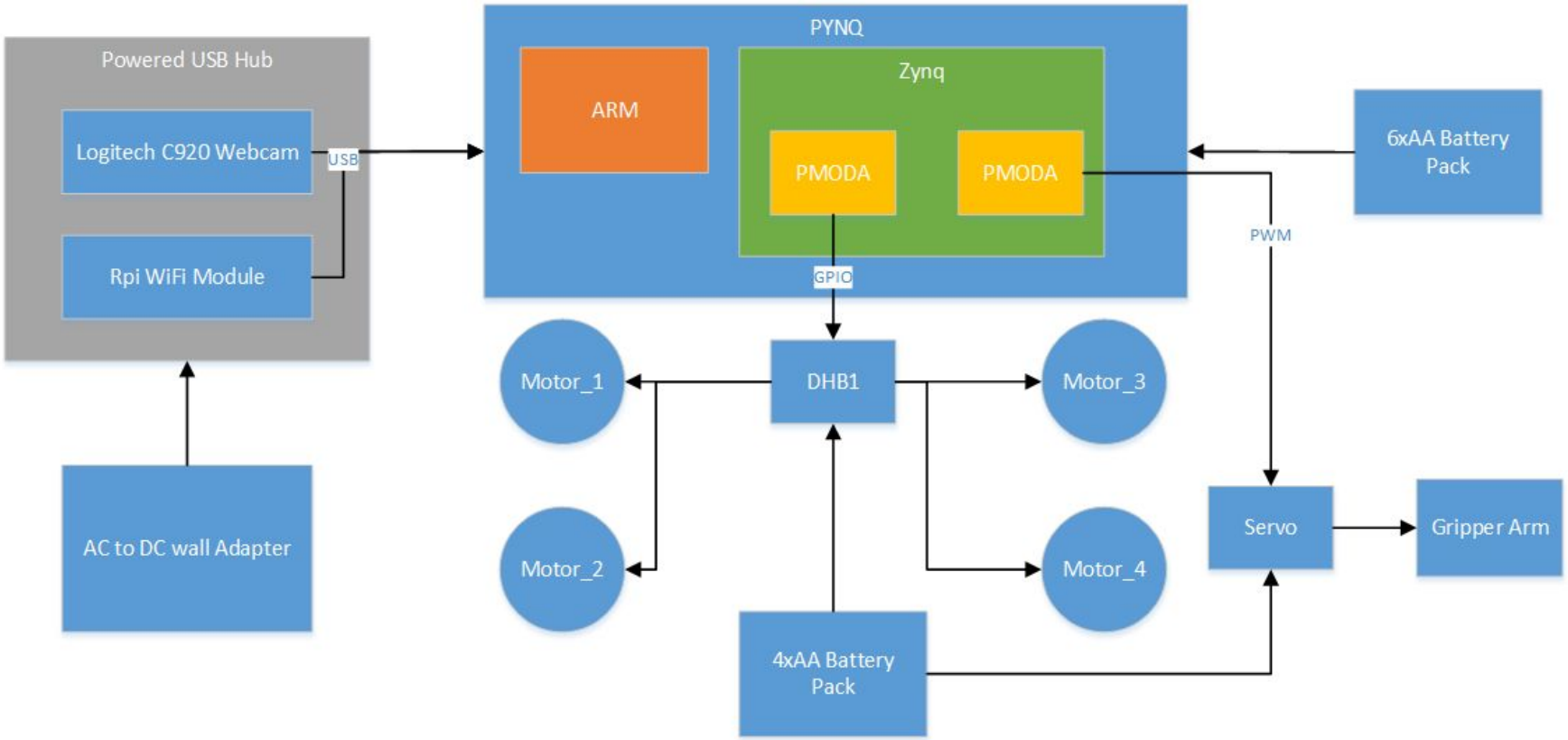
Logitech C920 USB Webcam: Used for image recognition and processing.

Wifi Dongle: Used for wireless control through Jupyter Notebook

DHB1: Used to control the robot motors

Actobotics Gripper Arm: Used for grabbing and moving a can.

Hardware Design



Implementation Options

- Custom Overlay:
  - An overlay is the Zynq design used to communicate to the ARM processor, including several MicroBlaze processors controlling the Arduino and PMOD IOP's (Input/Output Processors)
  - This is the most laborious and error prone as the base design is fairly complicated, allowing extensive configurability in software (aka C/Python)
- Custom IOP Software
  - Each IOP for the PMOD and Arduino shield allow each pin to be configured in a number of ways, allowing for I2C, SPI, PWM, GPIO, among other interface options without ever touching HDL.
  - This is a good middle ground, where the C code can be used to configure the IOP's to suit the user's need
- Custom Python Class
  - Using the existing C IOP designs, one can configure many applications without touching C-code or HDL
  - This allows the fastest development times, with the most restrictions.

Implementation Decision

- We investigated each implementation and these were our findings
  - The Custom Overlay required extensive gutting to replace the IOP functionality, or extensive time spent conforming to create a new IOP hardware configuration or extending current IOP designs, this was not feasible for the scope of this project.
  - The Custom IOP Software was evaluated in detail, and even attempted, but limitations in the interfaces and complications with the IOP interactions caused this path to be abandoned in favor for getting a working design implemented.
  - The Custom Python Code using existing IOP’s was ultimately the path we chose to go, deciding to utilize existing interfaces to create a custom robot controller that provided the functionality required for the project.

Robot Controller Class (Motors)

- DHB1:
  - Uses PMOD_IO example to treat Motor Enable and Direction and GPIO outputs
  - Required custom Python functions to define motor controls.
  - 2 motors were controlled by each h-bridge, the left 2 and the right 2 independently
  - Each function has 1 parameter, the duration of the activity. This is implemented using the time.sleep command that has ms granularity.
  - Interface:
    - **Forward:** Both sides enabled and direction set to forward
    - **Reverse:** Both sides enabled and direction set to reverse
    - **Left:** Both sides enabled, right set in forward, left set reverse
    - **Right:** Both sides enabled, right set in reverse, left set forward

Robot Controller Class (Gripper Arm)

- Servo:
  - Uses PMOD_PWM example to output a PWM signal to the Servo
  - Used period of 25ms
    - This gave a 0 degrees at 1% duty cycle for 500 microseconds for gripper closed/grab.
    - For an open state, the angle ratios of the gripper meant that a 3% or 750 microsecond pulse for gripper open/release.
- Findings:
  - More optimized designs were definitely possible if more time was alloted, utilizing custom HDL and specific peripheral IP blocks.
  - The high level customization features allowed for a quick and effective design to be implemented, allowing for focus off of robot interface and more on control and automation

Reference: http://www.robotshop.com/blog/en/arduino-5-minute-tutorials-lesson-5-servo-motors-3636

Computer Vision

Can Detection

Given 1 fixed camera, use computer vision techniques to generate:

- Object detection
- Characterization of the object of interest
- Approximate distance to object

Options Explored::

- Haar Cascade Classifier
- Feature Detection using an assortment of classic CV algorithms
- Deep Learning Network

CV development

Classic Cv approach:

- filter/smooth image
- edge detection
- contour identification & contour data (shape, length,etc)
- ROI generation
- Color masking and identification

Computer Vision

Distance Measurement

Triangle Similarity for object(to be detected) to the camera distance.

The width of the object(W) is measured and we set the object at a known_distance(D) from the camera.The apparent width(P) of the object is measured in pixels and the focal length(F) of the camera is calculated as follows:-

$F=(P \times D) / W$

After finding the focal length of the camera, the distance of camera from the object is calculated by similarity of triangles using:-

$D = (F \times W) / P$

Computer Vision

Color Detection

The algorithm for color detection of the can is done in Python using OpenCV implementation.The image comes from interfacing a webcam with the Pynq board. The image is taken and is converted to HSV scale.

The color detecting is done by basically defining the lower and upper range of the color to be detected on the HSV scale and masking it with the original image(to get only the color to be detected) and bit wising the mask and the original image .

Challenges ...

Haar Cascade Classifier

- Time debt
- Requires hours(days) of fine tuning
- Non trivial tool chain setup and usage
- Rotational invariance, illumination, noise has to be accounted for
- Easy to generate false positives
- Not a magic “wand”

Challenges ...

Integration

- Jupyter notebook interface and video stream peculiarities
- Intermittent wifi crashes
- Camera occlusion and pan tilt
- “TIME DEBT”

Challenges ...

Deep Learning Network

i) Neon:- Neon which is an open source python based DL framework working on linux environment was unable to be set up and get it running on Windows.

ii) Keras and Tensorflow:- Another DL framework for training the neural network had similar issues in being set up . The long time it took for setting up these tool chains slowed the progress reasonably and ultimately we had to backout from implementing our idea of training a neural network.

Challenges...

Physical Design:

i) **Power:** Isolation of motors and servos versus PYNQ and USB HUB. USB HUB with WiFi and WebCam required extra power provided by wall-wart.

ii) **Motor Quality:** At one point the motors stripped their hubs and required remedy. This was due to utilizing cheap hobby motors and could have been avoided if more robust motors were implemented.

iii) **Debug and Control:** The primary control method of the PYNQ is through a web-browser, which required WiFi connection. The WiFi module was spotty at best, and debug required the use of the UART serial port via USB, which was also the primary power supply, which required shuffling the power and data connections.

THANK YOU...