Frisbee Trajectory Tracking and Pole Detection Using Computer Vision - ABU ROBOCON 2017

Anmol Popli

Abstract:

This project involves tracking of flying Frisbee and pole detection for the problem statement of Robocon 2017 through computer vision techniques. Colour based detection using histogram and its advantages are discussed. Installation and configuration of OS in raspberry pi, OpenCV in visual studio and OpenMP is also discussed.

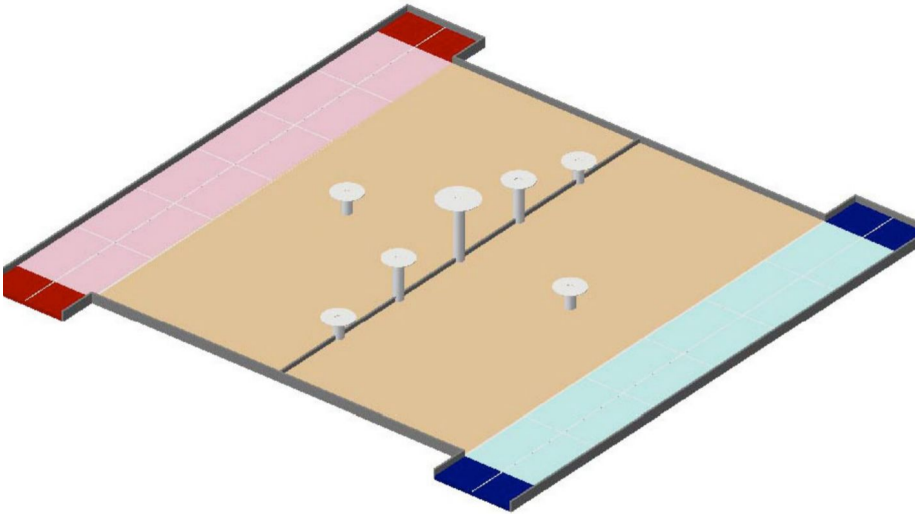

Figure 1: Robocon 2017 Game Field (Isometric View) - The pink and light blue regions are the Throwing Areas from where the robot launches frisbees so that they land on the poles

Softwares and Hardwares Used:

MATLAB
OpenCV (for windows in Visual Studio)
PiCam
Raspberry Pi
OpenMP

Install OpenCV in Visual Studio 2013:

i) Install Visual Studio 2013 and OpenCV 2.4.13 in directory C:Opencv_2.4.13.
ii) Set environment variables with following paths:
   a) C:\Opencv_2.4.13\build
   b) C:\Opencv_2.4.13\build\x64\vc12\bin

iii) Create a new empty project in Visual Studio. Go to Solution Explorer and right click on project name and then go to properties.

iv) If you use 64 bit OS, change platform to x64 in Configuration Manager.

v) Go to General in C/C++ option and set following paths:
   a) Additional Include Directories: C:\Opencv_2.4.13\build\include
   b) Additional #using Directories: $(OPENCV_BUILD)\x64\vc12\lib

vi) Go to General in Linker Option, change Additional Library Directories to C:\Opencv_2.4.13\build\x64\vc12\lib.

vii) Go to Input in Linker Option, copy and paste following path name in additional dependencies:

opencv_calib3d2413d.lib
opencv_contrib2413d.lib
opencv_core2413d.lib
opencv_features2d2413d.lib
opencv_flann2413d.lib
opencv_gpu2413d.lib
opencv_highgui2413d.lib
opencv_imgproc2413d.lib
opencv_legacy2413d.lib
opencv_ml2413d.lib
opencv_nonfree2413d.lib
opencv_objdetect2413d.lib
opencv_photo2413d.lib
opencv_stitching2413d.lib
opencv_ts2413d.lib
opencv_video2413d.lib
opencv_videostab2413d.lib
opencv_calib3d2413.lib
opencv_contrib2413.lib
opencv_core2413.lib
opencv_features2d2413.lib
opencv_flann2413.lib
opencv_gpu2413.lib
opencv_highgui2413.lib
opencv_imgproc2413.lib
opencv_legacy2413.lib
opencv_ml2413.lib
opencv_nonfree2413.lib
opencv_objdetect2413.lib
opencv_photo2413.lib
opencv_stitching2413.lib

opencv_ts2413.lib
opencv_video2413.lib
opencv_videostab2413.lib

Note: These steps are only valid in Visual Studio 2013 and OpenCV 2.4.13.

Install OS in Raspberry Pi:

Raspberry Pi's official OS is Raspbian. However, there are many third party Raspberry Pi OS like Ubuntu Mate, Windows 10 IOT Core, etc. Here, installation of Raspbian is presented. Raspbian has two version Raspbian Jessie With Pixel and Raspbian Jessie Lite. Both are based on Debian Jessie.
Raspbian Jessie lite is an minimal image of OS meaning that it uses less disk space. Jessie Lite also has less boot time that's why we used this OS in this project. However, installation procedure is same for both OS.

First download Jessie Lite from https://www.raspberrypi.org/downloads/raspbian/ and Win32DiskImager. Insert your SD card in laptop and then format. Now, write the downloaded Jessie Lite image to SD card. After completing this, Raspbian is installed.

Now, we will access raspberry pi remotely using SSH (Secure Shell which is a cryptographic network protocol used for securely connect to other network). But for this, you have to be on same network. We'll talk about it later. But, let's first enable SSH.

SSH can be enabled by two methods:

a) If you have access of Raspberry Pi:
   i) Enter sudo *raspi-config* in a terminal window.
   ii) Select *Interfacing Options*
   iii) Navigate to and select *SSH*
   iv) Choose *Yes*
b) For headless setup meaning you haven't access of raspberry pi, *SSH* can be enabled by placing a file named ssh, without any extension, onto the boot partition of the SD card. When the Pi boots, it looks for the ssh file. If it is found, *SSH* is enabled, and file is deleted. The content of the file does not matter: it could contain text or nothing at all.

Now, you have enabled SSH. To access pi, you should know IP address of Raspberry Pi. To set a static IP address to Raspberry Pi add statement 'ip=192.168.0.2' to cmdline.txt file in your SD card. Beware, there shouldn't be any space between ip and '=' sign. By the way, you can use any other IP.
If you use Windows, go to change adapter setting and then in Ethernet network properties option. First of all, you need to disable authentication for Ethernet and then set a static IPv4 address to Ethernet. As I have said before to

access PI over SSH, both devices should be on same network. So, you have to set IP address of Ethernet same as Pi Ip only changing last digit. In this case, IP address should be 192.168.0.3. Connect Raspberry Pi to laptop using Ethernet Cable. To use SSH, you have to install software which support SSH connection like Putty or MobaXterm. Putty doesn't support many things like SCP (Secure Copy which allows to transfer files between two devices connected using SSH). Also, MobaXterm has a nice GUI which helps to see file system in Raspberry Pi. So, I would recommend you to install MobaXterm over Putty. Now, open MobaXterm and click on 'Start New Terminal'. Use command 'ssh pi@ip_address' to connect to Raspberry Pi.
Now, first update OS using following commands:

c) sudo apt-get update
d) sudo apt-get upgrade

Add Wifi Network to Pi:

Open the file 'wpa_supplicant.conf' in '/etc/wpa_supplicant' directory. And paste the following lines below the file:

```
network={
        ssid="NameOfNetwork"
        psk="Password"
        key_mgmt=WPA-PSK
        priority=1
}
```

To connect the Raspberry by using this wifi. You need to connect to same network and know IP address of Pi which can be find out using software like Advanced IP Scanner or mobile application like Fing. Once, you know IP address of wifi, you can connect to Pi in same way as with Ethernet cable.

Install OpenCV with Python and C++:

Go to http://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/ and follow all the steps. OpenCV along with Python 2.7 and Python 3 will be installed.

Enable Raspberry Pi Camera:

To enable camera, go to http://pklab.net/?id=392&lang=EN and follow steps of installing Video4Linux only.

Install OpenMP:

OpenMP library is an open source library used to do multi-processing in devices. It can be used wherever a code can be split into two or more parallel program to save execution time. In this project, to increase frame rate we need

to capture frame as many possible. But if every image is processed, it'll take a lot of time, hence results in decreasing frame rate. Since, frame capturing doesn't require much processing power. It can be done on a single core. And rest three core can be used to process image. After frame capturing is done, all cores can be used to process image.

OpenMP comes preinstalled with C++ compiler. So, we don't need to install OpenMP, but to use this library it is needed to link this file to C++ file explicitly while compiling. For example, if you want to use this library in a C++ file named OpenMPTest.cpp, then you have to compile using following command:

g++ -o OpenMPTest OpenMPTest.cpp –fopenmp

But this can be frustrating. So, to get rid of this headache, CMake library (Cmake is an open-source, cross-platform family of tools designed to build, tests and package software) can be used. How it can be used is shown in below.

Install Raspberry Pi Cam Library:

Go to https://www.uco.es/investiga/grupos/ava/node/40 and follow all steps properly.

To use Raspberry Pi camera, we'll take help of CMake.

CMake:

Create a project and an empty text file with name CMakeLists.txt in it. Let's assume you want to execute file test.cpp

Copy and paste below code:

```
cmake_minimum_required (VERSION 2.8)
project (nameofproject)
include_directories("/home/pi/raspicam-0.0.5/src")
set(raspicam_DIR "/home/pi/raspicam-0.0.5/build")
find_package(raspicam REQUIRED)
find_package(OpenCV)
find_package(OpenMP)
IF  ( OpenCV_FOUND AND raspicam_CV_FOUND AND OPENMP_FOUND)
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${OpenMP_C_FLAGS}")
set (CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${OpenMP_CXX_FLAGS}")
MESSAGE(STATUS "COMPILING OPENCV TESTS")
add_executable (test test.cpp)
target_link_libraries (test ${raspicam_CV_LIBS})
ELSE()
MESSAGE(FATAL_ERROR "OPENCV NOT FOUND IN YOUR SYSTEM")
ENDIF()
```

Now execute following command (assuming you are in project directory):
mkdir build
cd build
cmake ..
make
./test

For more detail, refer to reference give below.

Color Thresholder App in Matlab:

This app was very useful in this project. It's a great tool to analyse image. Basically, it gives a nice user interface to see how image changes on applying threshold on different planes. It also shows histogram of image. It supports RGB, HSV, L*a*b, etc. formats.

Algorithm:

1) ***Colour Based Detection:***
   - ***i) Static Method:***

     Frisbee is detected using colour detection algorithm. First, image is converted to HSV format as it is less sensitive to light. Minimum and maximum thresholds are applied on HUE and VALUE plane.
   - ***ii) Dynamic or Adaptive Method:***

     Colour Thresholding method has a very big disadvantage. A very slight variation in light can change threshold value. To overcome this problem, a dynamic approach to find out threshold values using histogram is proposed here. This method can be applied under following assumptions:

     i) Approximate area of object is known.
     ii) One colour channel is sufficient to distinguish with other colour. For example, white colour detection can be detected only using HUE channel.
     iii) Maximum and minimum threshold is known.

     Steps to follow:

     i) First find out histogram of channel.
     ii) Start counting points from minimum index until points are smaller than area of object.
     iii) Apply this minimum and new threshold values to channel.

The main *ADVANTAGES* of this method are that it is background invariant and requires only one channel.

Procedure to detect poles and Frisbee using this method:

a) *Poles Detection*:

Poles are situated at known distances from camera. Their areas are fixed. Also, white colour poles detection require only HUE channel (in HSV format). Minimum and maximum thresholds are taken 0 and 150. A certain portion of poles have arena in background. So, algorithm is applied only to that portion.

b) Frisbee Detection:

Frisbee Colour is black. This is detected using VALUE plane. Although it sometimes requires two channel which can be converted to one channel by applying a fixed threshold on other channel.

2) Noise Filter:

To remove small size noises, we used *median filter* which takes median of aperture around a particular pixel of size given by user and assigns it to that pixel. We choose size of aperture to be "17, 7".

3) Frisbee Extraction:

Eccentricity, angle and area are used to distinguish the Frisbee from other objects. Since the Frisbee shape is like elliptical its eccentricity should be less than 1. A minimum threshold of 0.85 is applied on blobs. Also Frisbee was thrown vertically, its angle should be near to 0 or 180 degree. And then the blob which have closest area to the Frisbee of previous frame may be Frisbee.

But this result may be wrong as due to following reason:

i) If the Frisbee is in black background.

ii) Frisbee is not detected due to light illumination.

To check it is Frisbee or not, error between estimated position and predicted position (discussed below) of Frisbee is used. Now, if it comes out that the detected blob doesn't satisfy aforementioned condition, predicted position will be considered as Frisbee centroid.

4) Centroid Prediction:

Centroid is predicted using median of last four changes in coordinates of the Frisbee.

5) ***Multiprocessing in Raspberry Pi:***

For multiprocessing, OpenMP library is used. Raspberry Pi 3 has four cores, we use first core to take images and store them in memory to increase frame rate and all other three cores were used to do processing on images. Once, all frames are captured, all cores are used to process image.

6) ***Communication With Arduino:***

The communication protocol used to communicate Arduino and Pi is *UART.* I2C communication protocol can't be used as same protocol is used by Pi Cam. Also, on using SPI protocol USB remote stops working.

Important Functions:

There are three functions defined in FDetectionFunction.h file which are building blocks to implement above algorithm. These functions are useful in any object recognition problem. A brief description is given below:

a) *medFilt2():* This function is used to filter the noise using median of mask around a pixel. This function is defined in OpenCV library but it filters the image only using square shaped mask. But in our situation, we needed to use a rectangular mask as Frisbee was elliptical. This function can be implemented by counting number of zeroes in the mask around a pixel. If number of zeroes is less than number of pixel in mask than a value 1 will be assigned to this pixel else 0.

```
inline void medFilt2(Mat I, Mat I_median, int kernel[]){
        int ki = kernel[0] / 2;
        int kj = kernel[1] / 2;
        int kernelSize = (kernel[0] * kernel[1]) / 2;
        int* zero = new int[I.cols];
        Scalar temp;
        for (int i = 0; i < I.cols; i++)
                zero[i] = 0;

        //Store zero count of each column upto kernel[0] rows
        for (int i = 0; i < kernel[0]; i++){
                for (int j = 0; j < I.cols; j++){
                        temp = I.at<uchar>(i, j);
                        if ((int)temp.val[0] == 0)
                                zero[j]++;
                }
        }
```

```
        int zeroKernel = 0;
        for (int i = 0; i < kernel[1]; i++)
                zeroKernel += zero[i];

        for (int i = ki; i < I.rows - ki - 1; i++){
                for (int j = kj; j < I.cols - kj; j++){
                        if (j == kj)
                                I_median.at<uchar>(i, j) = (zeroKernel <=
kernelSize) ? 255 : 0;
                        else{
                                zeroKernel -= zero[j - kj - 1];
                                zeroKernel += zero[j + kj];
                                I_median.at<uchar>(i, j) = (zeroKernel <=
kernelSize) ? 255 : 0;
                        }
                }

                Scalar temp2;
                for (int k = 0; k < I.cols; k++){
                        temp = I.at<uchar>(i - ki, k);
                        temp2 = I.at<uchar>(i + ki + 1, k);
                        int value1 = ((int)temp.val[0] == 0) ? 1 : 0;
                        int value2 = ((int)temp2.val[0] == 0) ? 1 : 0;
                        zero[k] = zero[k] + value2 - value1;
                }

                zeroKernel = 0;
                for (int k = 0; k < kernel[1]; k++)
                        zeroKernel += zero[k];
        }
}
```

b) *mouseHandler():* This function is used to debug the code and analysis of image. It prints HSV and RGB values of a pixel when clicking on displayed image. This function is not directly called. It needs to be passed on an inbuilt function setMouseCallback which takes three input – name of window in which image is displayed, mouseHandler function and address of image. For example, If you want to print values of an image stored in frame Mat object displayed in window say “window”, use the statement:

```
        setMouseCallback("window", mouseHandler, &frame);
```

c) *getBlobProps():* This function is used to find out properties of blobs in binary image like contour areas, centroids, eccentricity, dimension, angle and contour itself. Inputs of this function are binary image and a struct which members are vector. After calling this function, contour properties will be stored in struct object.

```
struct Blob {
	vector<double> contourareas;
	vector<Point> centroids;
	vector<double> eccentricity;
	vector<vector<double> > dimension;
	vector<vector<Point> > contours;
	vector<float> angle;
};

inline void getBlobProps(Mat I, Blob* myBlobs)
{
	vector<Vec4i> hierarchy;
	findContours(I, myBlobs->contours, hierarchy, CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));

	for (int i = 0; i<(myBlobs->contours).size(); i++) {
		vector<Point> contour = (myBlobs->contours)[i];

(myBlobs->contourareas).push_back(contourArea(contour));
		Moments M = moments(contour);
		(myBlobs->centroids).push_back(Point(M.m10 / M.m00, M.m01 / M.m00));
		(myBlobs->dimension).push_back(*(new vector<double>));

		if ((myBlobs->contours)[i].size() > 5){
			RotatedRect ell = fitEllipse(Mat((myBlobs->contours)[i]));
			double a = ell.size.width / 2;	// width >= height
			double b = ell.size.height / 2;
			(myBlobs->dimension[(myBlobs->dimension).size() - 1]).push_back(ell.size.width);
			(myBlobs->dimension[(myBlobs->dimension).size() - 1]).push_back(ell.size.height);
			if (a > b)
				(myBlobs->eccentricity).push_back(0);
```

```
                    else
                        (myBlobs->eccentricity).push_back(sqrt(1 -
(a*a) / (b*b)));
                    (myBlobs->angle).push_back(ell.angle);
                }
                else{
                    RotatedRect rect = minAreaRect(Mat(contour));
                    (myBlobs->dimension[i]).push_back(rect.size.width);
                    (myBlobs->dimension[i]).push_back(rect.size.height);
                    //(myBlobs->eccentricity).push_back(sqrt(1 -
(rect.size.width*rect.size.width) / (rect.size.height*rect.size.height)));
                    (myBlobs->eccentricity).push_back(0);
                    (myBlobs->angle).push_back(rect.angle);
                }
        }
}
```

Note: For more detail, refer to FDetectionFunction.h file.

Future Work:

i) Dynamic method of colour detection can be improved for colours requiring more than two channels. For example, if RGB format is used, then to detect white colour all channels will be required but channel required can be made equal to 1 if HSV format is used.

ii) Kalman Filter based tracking can be applied for better results in the centroid prediction.

References:

1) Udacity Course: Introduction to Computer Vision
2) OpenCV: http://docs.opencv.org/master/d9/df8/tutorial_root.html
3) OpenMP: https://www.youtube.com/watch?v=nE-xN4Bf8XI&list=PLLX-Q6B8xqZ8n8bwjGdzBJ25X2utwnoEG
4) CMake: https://www.youtube.com/watch?v=6zOpxAwYKUQ
5) Configure Raspberry Pi: https://www.youtube.com/watch?v=T9Q4DUAmGG0