Tutorial on Object Tracking using a Pi-Cam interfaced on a Raspberry-Pi

e-Yantra Team

July 6, 2016

Contents

1 Object Tracking using a Pi-Cam interfaced on a Raspberry-Pi

Objective of this tutorial is to track a object from a given Region-Of-Interest and move the camera according to the object movement using servos.

2 Prerequisites

- Pan-Tilt Camera system interafced R-Pi
- Python and Open CV installed on the R-Pi

3 Hardware Requirement

- Raspberry-Pi B+ Development Board
- Raspberry-Pi Camera Module(Pi Camera)
- Ethernet Cable/Wifi Mdoule
- Camera mount frame

4 Software Requirement

- Python 2.7.11
- OpenCV 2.4.13
- numpy 1.7.1
- RPIO and picamera packages
- **Note :** These are the versions we were working on while creating this tutorial.

5 Theory and Description

Here We will be moving the camera according to the object movement in the frame so that the camera always tries to keep the object at its center.

To make the camera servos move in both Pitch and Roll directions at the same time simultaneously which is not possible as the servos don't have a feedback of how much distance they had moved or the angle they are in present state,and also we cant get a feedback through the center as both the camera and servos need the center at the same time for tracking and movement respectively.

This center controversy can be solved by multi-processing,in which we assign each Roll and Pitch servo a process in which they will be two global variables (one is Current Position of the servo and the second is Desired Position of the servo).These processes will be executing in the back-end all the time,but gets updated only when they get called by a function which updates the two global variables.

For the distance to be moved by the servos as we don't have a feedback from the servo,we can use the proportionality constant which can be obtained by calibrating the servos several times.
Note: We can also use PID controller for which we need to know the Integration and Derivation constants.

CAMShift tracking is done parallel to this camera movement in which again a center controversy may arise.Here as we know that the CAM shift works on the previous track window and previous center which will be a totally different co-ordinates when the camera moves frequently.

We can solve this as we know that camera always tries to keep the object at its center.So we gave the CAM shift its previous track window as the camera center.

Here issues may arise about "Is the servo fast enough to catch-up with the CAM shift track by keeping the object at its center every time?"

As we know the servos work in a totally different processes ,they are fast enough to meet up our needs of the CAM Shift track window (which we took a 20x20 square from the center).The implementation of this is shown in the below sections.

6 Experiment

The Python code is given below

```
#import necessary modules
import numpy as np
import cv2
import math
import functions
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
from RPIO import PWM
from multiprocessing import Process, Queue
```

```
#Initilaizing camera
camera = PiCamera()
camera.resolution = (320,240)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(320,240))
time.sleep(0.1)

##===================================================================================
#Initializing Servos
#servo GPIO connections
Roll = 23
Pitch = 24
# Upper limit
RollUL = 230
PitchUL = 230
# Lower Limit
RollLL = 60
PitchLL = 60
#initial Position
initroll = ((RollUL - RollLL) / 2) + RollLL
initpitch = ((PitchUL - PitchLL) / 2) + PitchLL
PWM.setup()
PWM.init_channel(0)
#init servos to center
PWM.add_channel_pulse(0, Roll, 0, initroll)
PWM.add_channel_pulse(0, Pitch, 0, initpitch)

RollCPQ = Queue()        # Servo zero current position, sent by subprocess
PitchCPQ = Queue()       # Servo one current position, sent by subprocess
RollDPQ = Queue()        # Servo zero desired position, sent by main and r
PitchDPQ = Queue()       # Servo one desired position, sent by main and re
RollSQ = Queue()         # Servo zero speed, sent by main and read by subp
PitchSQ = Queue()        # Servo one speed, sent by main and read by subpr
def P0():       # Process 0 controlles Pan servo
        speed = .1              # Here we set some defaults:
        RollCP = initroll - 1           # by making the current position
        RollDP = initroll               #       we can be sure we know wh

        while True:
                time.sleep(speed)
                if RollCPQ.empty():                     # Constantly upda
                        RollCPQ.put(RollCP)             #       to read i
                if not RollDPQ.empty():         # Constantly read read Ro
                RollDP = RollDPQ.get()   #        has updated it
        if not RollSQ.empty():                    # Constantly read
                RollS = RollSQ.get()     #        has updated it, t
                speed = .1 / RollS                #        the wait
        if RollCP < RollDP:                                        #
                RollCP += 1
                RollCPQ.put(RollCP)
               PWM.clear_channel_gpio(0, Roll)
               PWM.add_channel_pulse(0, Roll, 0, RollCP)
                if not RollCPQ.empty():                            #
                        trash = RollCPQ.get()
        if RollCP > RollDP:                                        #
                RollCP -= 1
                RollCPQ.put(RollCP)
               PWM.clear_channel_gpio(0,Roll)
               PWM.add_channel_pulse(0, Roll, 0, RollCP)
                if not RollCPQ.empty():                            #
                        trash = RollCPQ.get()
        if RollCP == RollDP:             # if all is good,-
                RollS = 1
# slow the speed; no need to eat CPU just waiting


def P1():        # Process 1 controlles Tilt servo using same logic as abo
        speed = .1
        PitchCP = initpitch - 1
        PitchDP = initpitch

        while True:
                time.sleep(speed)
                if PitchCPQ.empty():
                        PitchCPQ.put(Pitch)
                if not PitchDPQ.empty():
                        PitchDP = PitchDPQ.get()
                if not PitchSQ.empty():
                        PitchS = PitchSQ.get()
                        speed = .1 / PitchS
                if PitchCP < PitchDP:
                        PitchCP += 1
                        PitchCPQ.put(PitchCP)
                       PWM.clear_channel_gpio(0, Pitch)
                       PWM.add_channel_pulse(0, Pitch, 0,PitchCP)
                        if not PitchCPQ.empty():
                                trash = PitchCPQ.get()
        if PitchCP > PitchDP:
            PitchCP -= 1
            PitchCPQ.put(PitchCP)
            PWM.clear_channel_gpio(0, Pitch)
            PWM.add_channel_pulse(0, Pitch, 0, PitchCP)
            if not PitchCPQ.empty():
                trash = PitchCPQ.get()
        if PitchCP == PitchDP:
            PitchS = 1



Process(target=P0, args=()).start()      # Start the subprocesses
Process(target=P1, args=()).start()      #
time.sleep(1)                            # Wait for them to start
##==========================================================================


def CamRight( distance, speed ):                    # To move right, we are p
    global RollCP                        # We Global it so
everyone is on the same page about where the servo is...
    if not RollCPQ.empty():              # Read it's current position give
        RollCP = RollCPQ.get()           #       and set the main process
    RollDP = RollCP + distance           # The desired position is the cur
    if RollDP > RollUL:                  # But if you are told to move fur
        RollDP = RollUL                  # Only move AS far as the servo i
    RollDPQ.put(RollDP)                             # Send the new desired po
    RollSQ.put(speed)                               # Send the new speed to t
    return;

def CamLeft(distance, speed):                       # Same logic as above
    global RollCP
    if not RollCPQ.empty():
        RollCP = RollCPQ.get()
    RollDP = RollCP - distance
    if RollDP < RollLL:
        RollDP = RollLL
    RollDPQ.put(RollDP)
    RollSQ.put(speed)
    return;


def CamUp(distance, speed):                         # Same logic as above
    global PitchCP
```

```
        if not PitchCPQ.empty():
                PitchCP = PitchCPQ.get()
        PitchDP = PitchCP + distance
        if PitchDP > PitchUL:
                PitchDP = PitchUL
        PitchDPQ.put(PitchDP)
        PitchSQ.put(speed)
        return;


def CamDown(distance, speed):                          # Same logic as above
        global PitchCP
        if not PitchCPQ.empty():
                PitchCP = PitchCPQ.get()
        PitchDP = PitchCP - distance
        if PitchDP < PitchLL:
                PitchDP = PitchLL
        PitchDPQ.put(PitchDP)
        PitchSQ.put(speed)
        return;



#==========================================================================


#defining maximum possible shift in area and center admissible.
MAX_CENTER_SHIFT = 100
MAX_AREA_SHIFT = 5000
HUE_BIN = 180
SAT_BIN = 256

#This threshold will be used to not forget sliding window
THRESHOLD = 3

#Variable used to store standard deviation in histogram distance in previ
prev_standard_deviation = 0

#Variable is used to count number of frames
total_frames = 1

#This kernel will be used in erosion and dilation (opening)
kernel = np.ones((3,3),np.uint8)
```

```
#Flag to indicate first frame
first_frame = True

#Termination criteria for CAMShift to stop.
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )

#Window to show tracking
cv2.namedWindow('frame', cv2.WINDOW_NORMAL)



#Loop to get first bounding box
for image in camera.capture_continuous(rawCapture, format="bgr", use_vide

    frame= image.array

    #Show frame
    cv2.imshow('frame',frame)

    #Show some messages
    if first_frame:
        print 'press p to give roi'
        print 'press ESC to exit'
        first_frame = False

    k = cv2.waitKey(1)
    #If user presses p, get track window
    if k == ord('p'):
        (c,r,w,h) = functions.get_track_window(frame.copy())
        rawCapture.truncate(0)
        break

    #If user presses escape, exit program
    elif k == 27:
        rawCapture.truncate(0)
        cv2.destroyAllWindows()
        exit(0)
    rawCapture.truncate(0)
#print w*h
#print (c+w/2,r+h/2)

#Get height, width of the frame.
(height_frame,width_frame,channels) = frame.shape
```

```
#Change color space of frame from RGB to HSV
hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

#Use 50 pixels margin from bounding box for tracking
#Make white other area
if r > 50:
    hsv_frame[:r-50,:] = [255,255,255]
if r+h+50 < height_frame:
    hsv_frame[r+h+50:,:] = [255,255,255]
if c > 50:
    hsv_frame[:,:c-50] = [255,255,255]
if c+w+50 < width_frame:
    hsv_frame[:,c+w+50:] = [255,255,255]

#Mask frame for better result
mask_frame = cv2.inRange(hsv_frame, np.array((0., 60.,32.)), np.array((18

#Get Region of interest according to bounding box
hsv_roi = hsv_frame[r:r+h, c:c+w]
mask_roi = mask_frame[r:r+h, c:c+w]

#Get 2D roi histogram and normalize it
hist_roi = cv2.calcHist([hsv_roi],[0,1],mask_roi,[HUE_BIN,SAT_BIN],[0,180
cv2.normalize(hist_roi,hist_roi,0,255,cv2.NORM_MINMAX)

#Get 2D frame histogram and normalize it
hist_frame = cv2.calcHist([hsv_frame],[0,1],mask_frame,[HUE_BIN,SAT_BIN],
cv2.normalize(hist_frame,hist_frame,0,255,cv2.NORM_MINMAX)

#Get mean of histogram distance
prev_mean = cv2.compareHist(hist_roi,hist_frame,method=cv2.cv.CV_COMP_BHAT

#Get ROI back projection on frame
back_projection = cv2.calcBackProject([hsv_frame],[0,1],hist_roi,[0,180,0

#Get track window and apply CAMShift
track_window = (c,r,w,h)
retval, track_window = cv2.CamShift(back_projection, track_window, term_c
(c,r,w,h) = track_window
Center=[c+w/2,r+h/2]
if Center[0] != 0:             # if the Center of the frame is not zero

    if Center[0] > 180: # The camera is moved diffrent distances and spee
        CamRight(1,1)
    if Center[0] > 190: #
        CamRight(2,2)    #
    if Center[0] > 200: #
        CamRight(6,3)
    if Center[0] > 230: #
        CamRight(8,3)      #

    if Center[0] < 140: # and diffrent dirrections depending on what side
        CamLeft(1,1)
    if Center[0] < 130:
        CamLeft(2,2)
    if Center[0] < 120:
        CamLeft(6,3)
    if Center[0] < 90:  #
        CamLeft(8,3)

    if Center[1] > 140: # and moves diffrent servos depending on what axi
        CamUp(1,1)
    if Center[1] > 150:
        CamUp(2,2)
    if Center[1] > 160:
        CamUp(6,3)
    if Center[1] >190:
        CamUp(8,3)

    if Center[1] < 100:
        CamDown(1,1)
    if Center[1] < 90:
        CamDown(2,2)
    if Center[1] < 80:
        CamDown(6,3)
    if Center[1] < 50:
        CamDown(8,3)

(c,r,w,h) = (140,100,40,40)
track_window = (140,100,40,40)
prev_area = 40*40
prev_center = (160,120)
#print '******'
#print prev_center
#print prev_area
#print prev_mean
#print '******'
```

```
#Flag to get whether roi is lost or not.
isLost = False

#Main loop

for image in camera.capture_continuous(rawCapture, format="bgr", use_vide

    frame= image.array
    #Change color space of frame from RGB to HSV
    hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    #If roi is not lost, Track ROI
    if not isLost:
        #Increment frame number
        total_frames = total_frames + 1

        #Use 50 pixels margin from bounding box for tracking
        #Make white other area
        if r > 50:
            hsv_frame[:r-50,:] = [255,255,255]
        if r+h+50 < height_frame:
            hsv_frame[r+h+50:,:] = [255,255,255]
        if c > 50:
            hsv_frame[:,:c-50] = [255,255,255]
        if c+w+50 < width_frame:
            hsv_frame[:,c+w+50:] = [255,255,255]

        #Mask frame for better result
        mask_frame = cv2.inRange(hsv_frame, np.array((0., 60.,32.)), np.a

        #Get 2D frame histogram and normalize it
        hist_frame = cv2.calcHist([hsv_frame],[0,1],mask_frame,[HUE_BIN,S
        cv2.normalize(hist_frame,hist_frame,0,255,cv2.NORM_MINMAX)

        #Get histogram similarity between rame and ROI
        histogram_distance = cv2.compareHist(hist_roi,hist_frame,method=c
        current_mean = prev_mean + ((histogram_distance - prev_mean) / to

        #Get Standard Deviation of Histogram distance
        current_standard_deviation = math.sqrt((((total_frames-2) * (prev

        #Threshold for Histogram Distance
        adaptive_threshold = (current_mean + THRESHOLD * current_standard
        #print 'histogram_distance = '
        #print histogram_distance
        #print 'adaptive_thes = '
        #print adaptive_threshold

        #ROI lost if
        if histogram_distance > adaptive_threshold :
            isLost = True
            print 'lost adaptive'

        else:
            #Circulating variables
            prev_mean = current_mean
            prev_standard_deviation = current_standard_deviation

            #Get ROI back projection on frame
            back_projection = cv2.calcBackProject([hsv_frame],[0,1],hist_

            #Apply CAMShift
            retval, track_window = cv2.CamShift(back_projection, track_wi
            (c,r,w,h) = track_window

            if c<=0 or r<=0 or w<= 0 or h<=0:
                    isLost = True
                    print 'lost due to track_window'

            else:
                    Center = (c+w/2,r+h/2)

                    cv2.circle(frame,Center,6,(255,0,255),-1)

                    if Center[0] != 0:                # if the Center of th

                            if Center[0] > 180: # The camera is moved dif
                                CamRight(1,1)
                            if Center[0] > 190: #
                                CamRight(2,2)     #
                            if Center[0] > 200: #
                                CamRight(6,3)
                            if Center[0] > 230: #
                                CamRight(8,3)      #

                            if Center[0] < 140: # and diffrent dirrection
                                CamLeft(1,1)
                                if Center[0] < 130:
                                    CamLeft(2,2)
                                if Center[0] < 120:
                                    CamLeft(6,3)
                                if Center[0] < 90:  #
                                    CamLeft(8,3)

                                if Center[1] > 140: # and moves diffrent serv
                                    CamUp(1,1)
                                if Center[1] > 150:
                                    CamUp(2,2)
                                if Center[1] > 160:
                                    CamUp(6,3)
                                if Center[1] >190:
                                    CamUp(8,3)

                                if Center[1] < 100:
                                    CamDown(1,1)
                                if Center[1] < 90:
                                    CamDown(2,2)
                                if Center[1] < 80:
                                    CamDown(6,3)
                                if Center[1] < 50:
                                    CamDown(8,3)
                        (c,r,w,h) = (140,100,40,40)
                        track_window = (140,100,40,40)
                        prev_area = 40*40
                        prev_center = (160,120)

        #Procedure to re-recognize the object
        else:
            #Get ROI back projection on frame
            back_projection1 = cv2.calcBackProject([hsv_frame],[0,1],hist_roi

            #back_projection1 = cv2.GaussianBlur(back_projection,(5,5),0)

            #Create binary image using OTSU algorithm
            ret3,back_projection1 = cv2.threshold(back_projection1,0,255,cv2.

            #Apply erosion and dilation
            back_projection1 = cv2.morphologyEx(back_projection1, cv2.MORPH_C


            #Get contours
```

```
contours, hierarchy = cv2.findContours(back_projection1,cv2.RETR_

#Get good contours
good = []
for i in range(len(contours)):
    #area = cv2.contourArea(contours[i])
    #if area > 0.1 * prev_area:
    good.append(i)

'''
#if there is only one contour that is ROI
if len(good) == 1:
    print 'a'
    c,r,w,h = cv2.boundingRect(contours[good[0]])
    track_window = (c,r,w,h)
    prev_area = w*h
    prev_center = (c+w/2,r+h/2)
    if c > 0 and r > 0 and w > 0 and h > 0:
        isLost = False
        print 'found'
'''

#If some good contours found
if len(good) > 0:
    print 'b'
    #print len(good)
    #print 'some good contours found'

    #List for bounding box
    bounding_box = [[0]*4]*len(contours)

    #List for histogram distance
    hist_dist = [60000]*len(contours)

    #Loop to get histogram distance of each good contour from roi
    for i in good:
        bounding_box[i][0],bounding_box[i][1],bounding_box[i][2],
        roi_detected = hsv_frame[bounding_box[i][1]:bounding_box[
        mask_roi_detected = cv2.inRange(roi_detected, np.array((0
        hist_roi_detected = cv2.calcHist([roi_detected],[0,1],ma
        cv2.normalize(hist_roi_detected,hist_roi_detected,0,255,c
        hist_dist[i] = cv2.compareHist(hist_roi,hist_roi_detected

    #Get closest histogram distance
```

```
min_hist_dist = min(hist_dist)

#Get track window if
if min_hist_dist < adaptive_threshold:
    i = hist_dist.index(min_hist_dist)
    c = bounding_box[i][0]
    r = bounding_box[i][1]
    w = bounding_box[i][2]
    h = bounding_box[i][3]
    track_window = (c,r,w,h)

    Center = (c+w/2,r+h/2)

    if Center[0] != 0:                # if the Center of the fa

        if Center[0] > 180: # The camera is moved diffrent di
            CamRight(1,1)
        if Center[0] > 190: #
            CamRight(2,2)   #
        if Center[0] > 200: #
            CamRight(6,3)
        if Center[0] > 230: #
            CamRight(8,3)    #

        if Center[0] < 140: # and diffrent dirrections depend
            CamLeft(1,1)
        if Center[0] < 130:
            CamLeft(2,2)
        if Center[0] < 120:
            CamLeft(6,3)
        if Center[0] < 90:  #
            CamLeft(8,3)

        if Center[1] > 140: # and moves diffrent servos depen
            CamUp(1,1)
        if Center[1] > 150:
            CamUp(2,2)
        if Center[1] > 160:
            CamUp(6,3)
        if Center[1] >190:
            CamUp(8,3)

        if Center[1] < 100:
            CamDown(1,1)
```

```
                    if Center[1] < 90:
                        CamDown(2,2)
                    if Center[1] < 80:
                        CamDown(6,3)
                    if Center[1] < 50:
                        CamDown(8,3)
                (c,r,w,h) = (140,100,40,40)
                track_window = (140,100,40,40)
                prev_area = 40*40
                prev_center = (160,120)


                #If valid bounding box
                if c > 0 and r > 0 and w > 0 and h > 0:
                    isLost = False
                    print 'found'
    print str(Center[0]) + "," + str(Center[1])
    #Show frame
    cv2.imshow('frame',frame)

    rawCapture.truncate(0)
    if cv2.waitKey(1) & 0xff == 27: #ascii value for escape key
        break

#Release Servos
PWM.clear_channel(0)
PWM.cleanup()
cv2.destroyAllWindows()
```

7 References

The idea of using multi-processing is taken from ServoBlaster package used in R-Pi for multiple servo controlling written by Richard Hirst.

1. `https://github.com/richardghirst/PiBits/tree/master/ServoBlaster`

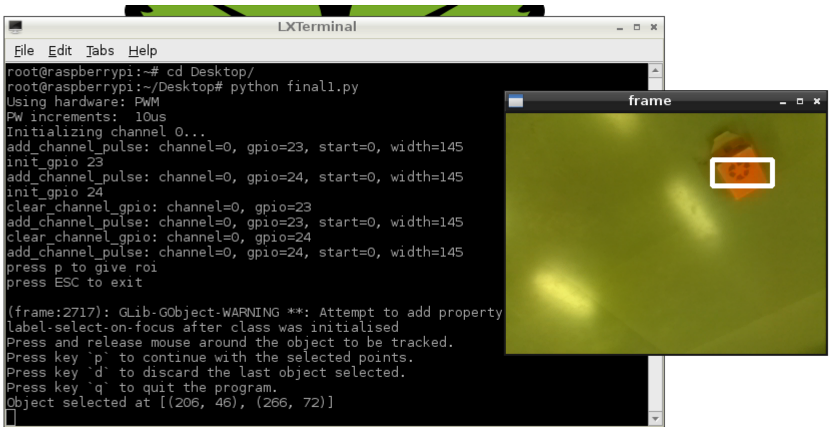
2. `https://www.raspberrypi.org/forums/viewtopic.php?f=44&t=54067`

3. `https://www.youtube.com/watch?v=m99n2heDXE87`

4. `http://www.instructables.com/id/Raspberry-Pi-Ball-tracking/`

8 Exercise

Object tracking with camera movement is shown below.



```
root@raspberrypi:~# cd Desktop/
root@raspberrypi:~/Desktop# python final1.py
Using hardware: PWM
PW increments:  10us
Initializing channel 0...
add_channel_pulse: channel=0, gpio=23, start=0, width=145
init_gpio 23
add_channel_pulse: channel=0, gpio=24, start=0, width=145
init_gpio 24
clear_channel_gpio: channel=0, gpio=23
add_channel_pulse: channel=0, gpio=23, start=0, width=145
clear_channel_gpio: channel=0, gpio=24
add_channel_pulse: channel=0, gpio=24, start=0, width=145
press p to give roi
press ESC to exit

(frame:2717): GLib-GObject-WARNING **: Attempt to add property
label-select-on-focus after class was initialised
Press and release mouse around the object to be tracked.
Press key `p` to continue with the selected points.
Press key `d` to discard the last object selected.
Press key `q` to quit the program.
Object selected at [(206, 46), (266, 72)]
```

Figure 1: Selction of object through ROI

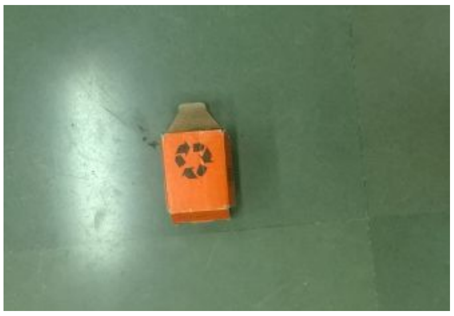


Figure 2: Initial position of the object

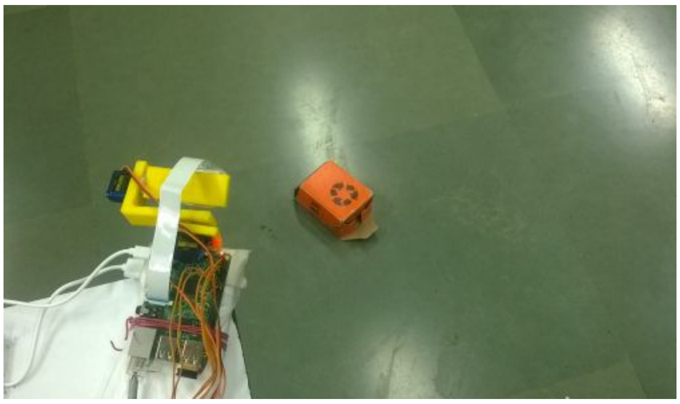
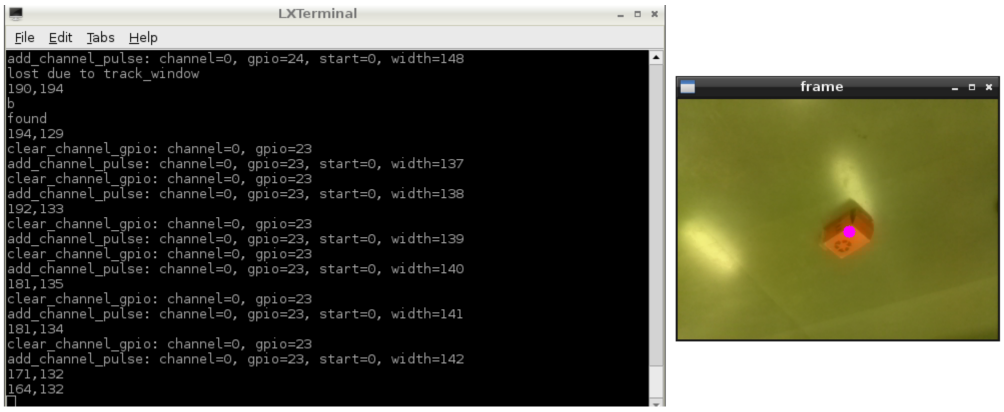


Figure 3: Object slightly moved to left



```
add_channel_pulse: channel=0, gpio=24, start=0, width=148
lost due to track_window
190,194
b
found
194,129
clear_channel_gpio: channel=0, gpio=23
add_channel_pulse: channel=0, gpio=23, start=0, width=137
clear_channel_gpio: channel=0, gpio=23
add_channel_pulse: channel=0, gpio=23, start=0, width=138
192,133
clear_channel_gpio: channel=0, gpio=23
add_channel_pulse: channel=0, gpio=23, start=0, width=139
clear_channel_gpio: channel=0, gpio=23
add_channel_pulse: channel=0, gpio=23, start=0, width=140
181,135
clear_channel_gpio: channel=0, gpio=23
add_channel_pulse: channel=0, gpio=23, start=0, width=141
181,134
clear_channel_gpio: channel=0, gpio=23
add_channel_pulse: channel=0, gpio=23, start=0, width=142
171,132
164,132
```

Figure 4: Camera also slightly moved to left

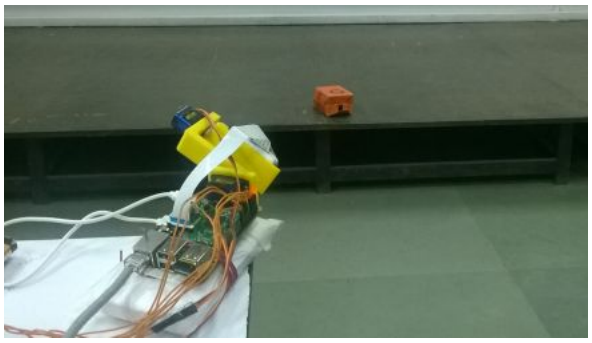
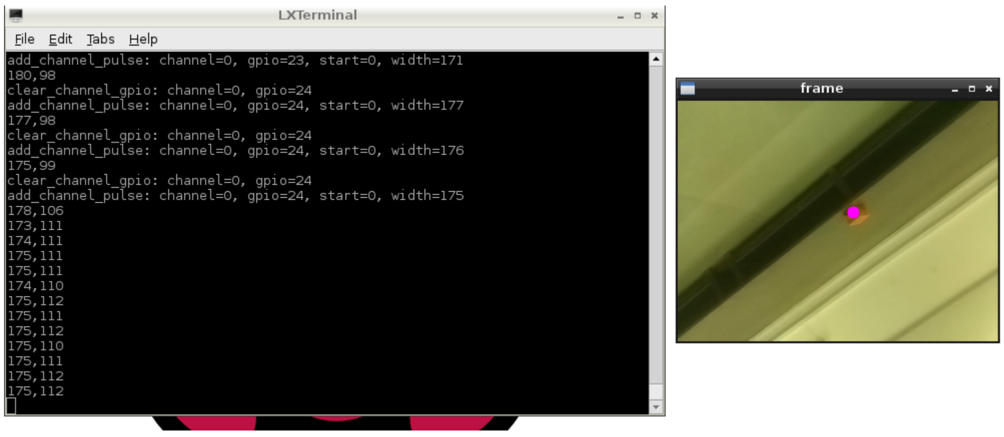


Figure 5: Object largely deviated



```
add_channel_pulse: channel=0, gpio=23, start=0, width=171
180,98
clear_channel_gpio: channel=0, gpio=24
add_channel_pulse: channel=0, gpio=24, start=0, width=177
177,98
clear_channel_gpio: channel=0, gpio=24
add_channel_pulse: channel=0, gpio=24, start=0, width=176
175,99
clear_channel_gpio: channel=0, gpio=24
add_channel_pulse: channel=0, gpio=24, start=0, width=175
178,106
173,111
174,111
175,111
175,111
174,110
175,112
175,111
175,112
175,110
175,111
175,112
175,112
```

Figure 6: Camera sucessfully catching up with the object