UAV Pursuit of a Moving Target

Jacqui Abalo[1], Andrew Klingelhofer[2], and John Ryan[3]

***Abstract*— We implement the Kalman Filter in a project which combines recent theory in object tracking and obstacle recognition with modern drone technology (Parrot Bebop 2.0) to perform UAV pursuit of a moving target. The Filter processes keypoints and candidate areas in an input frame recorded by the onboard camera, and outputs an estimation of the location of a pre-selected object. Our project does not use learning, so the UAV may be programmed to follow an arbitrary object in only one initial frame. Furthermore, the Filter is applied to process obstacles, and track their movement across several frames. This project is a test of the Kalman Filter's ability to handle noise as found in drone flight instability, keypoint misidentification, and sudden changes in camera orientation.**

I. BACKGROUND

Applications of a software by which a UAV may autonomously pursue a moving target abound: examples include video capture for demonstrative purposes, police pursuit of an evading suspect, and monitoring animal behavior. Due to advances both in drone technology and computer vision techniques in the last few years, this area is exciting and worthy of research and experimentation.

II. PROBLEM IDENTIFICATION

To what extent can current object recognition techniques coupled with adaptive modifications allow a modern multicopter to follow a moving target? Our problem is to explore this question by attempting to program a Parrot Bebop drone to avoid obstacles and to follow a person wearing a T-shirt with a distinct symbol on the back as the person walks around.

III. PROJECT AIM

The initial aim of our project was using the imaging capability of our Parrot Bebop Drone to create an algorithm using a Kalman filter [3] that recognizes an object and follows said object while avoiding other obstacles. We slightly modified by removing the obstacle avoidance. Our aim now reads, we aim to use the imaging capability of our Parrot Bebop Drone to create an algorithm using a Kalman filter [3] that recognizes an object and follows it. The Kalman Filter allows us to identify key points or points of interest on the object, providing information about the object needed to continually identify and update positioning. From that information, we create a bounding box as shown in the results section with various key points and estimations of the center of the object. This is particularly useful when computing the drones distance from the object. We are limited by the resolution of the drones camera in that we are unable to identify the object when too close or too far way. This bounding box and estimated center will provide a comprehensive way to compute the necessary distance the drone needs to be in order to correctly identify the object. Heres a high level description of the steps of our process:

1) Drone captures video image (.h264 file)
2) Convert .h264 file to mp4.
3) Send .mp4 to CMT and receive bounding box information.
4) Pass CMT data to Kalman Filter for noise reduction processing.
5) Kalman Filter identifies key points and creates bounding box.
6) Based on the information we receive from the Kalman Filter, send commands to the drone in order to keep within range of object.
7) Repeat.

IV. RESEARCH QUESTIONS

In researching libraries for the Parrot Bebop Drone, we found two, [4] and [5]. One is a node.js library, the other is Python. We found the node.js library more comprehensive and usable, so we have chosen to use that one as our functioning library. Further code details on our implmentation are given in Appendix A.

With both of these libraries, we are able to control the drone using either Python for [4] or javascript for

[1]Jacqui Abalo is a sophomore majoring in Computer Science at NYU Abu Dhabi

[2]Andrew Klingelhofer is a graduating senior in Gallatin studying Computer Science and the social, political, and economic effects of technology.

[3]John Ryan is a graduating senior in CAS majoring in Computer Science and Mathematics.

[5]. They provide us with the means to easily add drone flying functionality to our already existing Kalman filter.

Our initial research questions were as follows:

1) How can we use the data we gather from the Kalman filter to help us avoid obstacles while keeping pace with the target object?
2) What limitations or advantages are encountered when planning a path with a drone? How might those differences inform our understanding of path planning?

However, due to time complications, our research questions were adjusted to the following:

1) How effective is a Kalman filter at handling noise associated with target tracking by a drone?
2) What are the limitations to real-time object-tracking by unmanned vehicles as presented by this project?

V. SIGNIFICANCE OF RESEARCH QUESTIONS

1) **How effective is a Kalman filter at handling noise associated with target tracking by a drone?** - Said noise could occur in the form of flight instability, keypoint misidentification, sudden changes in camera orientation, or target obstruction. The importance of the Kalman filter is that it allows the target to be continuously tracked without any interruptions caused by noise or obstructions. Its efficacy would affect how well an object can be tracked.
2) **What are the limitations to real-time object-tracking by unmanned vehicles as presented by this project?** - As the scope of our research project is currently narrow, this is an important question to answer for the purposes of project expansion. That is to say, if this project were taken further and implemented on a larger scale, what bottlenecks would we have to overcome?

VI. LITERATURE REVIEW

1) **Computer Vision Based General Object Following for GPS Denied Multirotor Unmanned Vehicles**: This is a project quite similar to ours, except that it uses an AR Drone 2.0 with the OpenTLD library (as compared to our Parrot Bebop with the CMT library) to autonomously detect and follow a variety of objects at varying distances. Though the scope of this project is wider than ours, it acts as a useful guideline.
2) **Clustering of Static Adaptive Correspondences for Deformable Object Tracking (CMT)**: CMT is an algorithm which, given an initial image of the target object, tracks the object in a series of frames. The algorithm is discussed in more detail in the theoretical framework section.
3) **Object Tracking Using a Kalman Filter**: This MATLAB implementation of a Kalman filter for face tracking is the basis for our python implementation. As is the case with our project, the state refers to the bounding box of the detected face, given by the upper left and lower right corners of the box.
4) **Robotica/Katarina**: A python library for controlling the Parrot Bebop drone.
5) **Hybridgroup/node-bebop**: A node.js library for controlling the Parrot Bebop drone.

VII. THEORETICAL FRAMEWORK

1) **Object recognition in a moving frame using CMT**: CMT is the method used for detecting the target object. The algorithm works as follows: Initially, the target object is broken down into various keypoints, each with its own descriptor. Keypoints are tracked from previous frame to current frame by estimating optic flow. Keypoints are then matched by comparing their descriptors. The errors associated with tracking and matching are reduced by letting each keypoint vote for the target objects center. Votes are clustered and outliers are removed. Then a new bounding box is computed based on remaining keypoints.
2) **Kalman Filtering**: As explained by Welch and Bishop in [6], a Kalman Filter is defined by a set of mathematical equations that provide a recursive means of estimating the state of a process in a way that minimizes error, allowing for the estimation of past, present and future states. For the purposes of our project, the state refers to the location of the target object, which is defined by a bounding box. Our Kalman filter estimates the location and dimensions of this bounding box on the screen, using the coordinates of the upper left and lower right corners.

VIII. RESEARCH METHODS/METHODOLOGY

Our methods followed a gradual approach to the final goal. We started by examining existing real time object-recognition libraries to learn about current methods in

tracking objects in video, and settled on using CMT for this research project. We then moved on to testing software in various settings and with various different objects - first with a stationary camera, then with a hovering UAV, and finally with a mobile drone.

IX. RESULTS

Our first goal was to achieve object tracking via a modern algorithm. Using the CMT algorithm outlined in [1], we track noisy (having many keypoints) objects as they move around in a video. The following are a few screen shots to exhibit this:

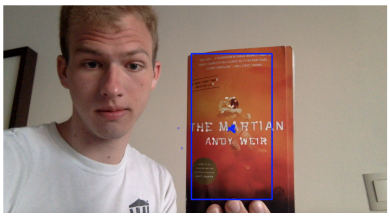

Fig. 1. Keypoints are designated by blue and white dots.

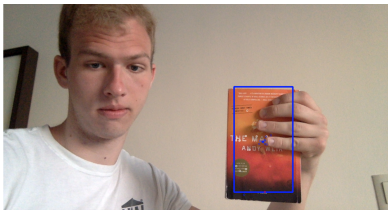

Fig. 2. The program is adaptive to partial (but not complete) obstructions.

Fig. 3. Recognition is not hindered by change in scale.

We then fed the coordinates of the blue rectangle to a Kalman Filter, so as to handle observation and process noise, and to allow tracking to persist through complete obstruction. The following screen shots demonstrate the effect of this addition.

Fig. 4. Blue is initial approximation, green is Kalman Filtered approximation.

Fig. 5. The Kalman Filter allows an approximation to remain even when the object is completely obstructed.

Finally, we applied this procedure to video obtained from the Parrot Bebop multicopter. As a physical demonstration of the object tracking, we've included instructions in the program to detect the horizontal displacement of the object from the center of the frame, and to direct the drone to fly to a different spatial/angular configuration such that the object will return to the center of the frame. The following images demonstrate the drone following a distinct logo on a sweater as its moved around a room.

Fig. 6. Immediately after takeoff, the drone takes a photo and the user specifies the target.

Fig. 7. Applying the Kalman Filter to the results of the object tracking algorithm, the drone rotates to keep the object in view.

Due to issues with processing speed, our current drone implementation suffers from a time lag, meaning that a fast-moving target cannot be tracked effectively.

X. CONCLUSIONS

We conclude that the Kalman Filter has worked in reducing the noise and producing a smooth output based on the noisy output of the object tracking algorithm. Furthermore, it successfully dealt with the noise of the video signal from the multicopter, which existed largely due to the instability and persistent self-corrective maneuvers of the multicopter in flight.

Although our implementation involves only slightly modifying the drone's configuration to keep an object in flight, future work may go further with this task, and consider changes in altitude of the target, as well as permanent obstructions around which the drone must navigate. The effectiveness of the Kalman Filter in our implementation should give confidence that other navigation algorithms can be applied effectively in such endeavors.

XI. RESOURCES

1) Parrot Bebop Drone (with built-in camera)
2) Python
3) Hybridgroup/node-bebop - details found in [5]
4) OpenCV/CMT

REFERENCES

[1] Nebehay, G. and Pflugfelder, R. (2015). Clustering of Static-Adaptive Correspondences for Deformable Object Tracking. http://www.gnebehay.com/publications/cvpr_2015/cvpr_2015.pdf.

[2] Pestana, S. L. and Saripalli, C. (2014). Computer Vision Based General Object Following for GPS denied Multirotor Unmanned Vehicles. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6858831.

[3] (2011). Object Tracking Using a Kalman Filter(MATLAB). https://blog.cordiner.net/2011/05/03/object-tracking-using-a-kalman-filter-matlab/.

[4] Robotika/katarina. (2015). https://github.com/robotika/katarina.

[5] Hybridgroup/node-bebop.https://github.com/hybridgroup/node-bebop.

[6] Welch, G. and Bishop, G. (2006). An introduction the Kalman Filter. https://s3.amazonaws.com/piazza-resources/ij4dty42mbl4dc/imf3e5rk2on5le/welchbishopkalman_intro.pdf?AWSAccessKeyId=AKIAIEDNRLJ4AZKBW6HA&Expires=1462244427&Signature=PRnipFE3ef%2Bb%2FshUJRnZCvRejt0%3D.

XII. APPENDIX

[A] - Detailed Description of Implementation

As mentioned in Section IV, we use a node.js library for communication with the drone (see [5]). Using this library, we are able to stream video from the drone's camera to a h264 file. This works in real time (other methods involve storing video inside the drone's memory for later retrieval) and allows for commands to be sent to the drone simultaneously. However, in order to use OpenCV to do video processing for object recognition, we need the video file in a different format. To handle this, we use Python's *subprocess* module to execute the bash command *ffmpeg* to convert from h264 to mp4.

Moreover, we don't run *ffmpeg* on the video.h264 file which is constantly being updated, because that would result in several redundant conversions, and a collapse in performance after even a short period of time. To work around this, we employ the Python *read* method for files, to continually check the video.h264 file for more bytes. When new bytes were found, they are written to a new file, temp.h264. Then, *ffmpeg* converts these files to mp4 files, which are then processed by OpenCV for object tracking.

There are several problems with this approach. Continually performing file opening and closing is horrible for real time performance. *ffmpeg* fails to convert a stream of bytes from h264 to mp4 from time to time. The h264 stream from the drone via the node.js library is very choppy. We encourage readers interested in trying a similar project to begin with getting clean and quick video stream from the drone's camera to the device performing the object recognition computations.

For each frame that CMT processes, a CSV file named after the frame number is written. This CSV file contains, in clockwise order beginning from the top left corner, the coordinates of each corner of the bounding box computed by CMT. A png image showing this CMT-measured bounding box in blue is also saved to the output directory. Kalman filtering relies on the data in the CSV file, reading in the top left and bottom right corners to represent the state of the object on the screen as measured by CMT. Filtering is performed on this measured state to provide the estimated state. For comparison purposes, a new green bounding box using the estimated state as coordinates is also drawn on the png image showing the CMT-measured state.

Finally, once our Python program has identified the location and size of the object's bounding box as approximated by the Kalman Filter, it decides whether to give instructions to the drone. For example, if the box has shifted, the drone should rotate. If the box has changed size, the drone should move forward/backward. If the box has done both simultaneously, the drone should drift either left, right, up, or down. To communicate the instruction to the node.js file, the Python process writes a number to a file in the same directory. Once a second, the node.js process checks this file, and sends a command to the drone based on what number is written. Any number read from the file is then set back to 0 (meaning hover), unless the number was 2 (meaning land).

[B] - Python Implementation of [3]. Kalman filter and object tracking.

```
import cv2
import numpy
import os.path
import numpy.matlib
from numpy.linalg import inv

def kalman_predict(x,P,F,Q):
    x = F*x;         #predicted state
    P = F*P*F.transpose() + Q; #predicted estimate covariance
    res = [x, P];
    return res;

def kalman_update(x,P,z,H,R):
    y = z - H*x;     #measurement error/innovation
    S = H*P*H.transpose() + R; #measurement error/innovation covariance
```

```
    K = P*H.transpose()*inv(S); #optimal Kalman gain
    x = x + K*y; #updated state estimate
    P = (numpy.matlib.identity(x.shape[0]) - K*H)*P; #updated estimate

    res = [x, P];
    return res;

# define the filter

x = numpy.matlib.zeros((6,1));
F = numpy.matrix('1 0 0 0 1 0;
                  0 1 0 0 0 1;
                  0 0 1 0 1 0;
                  0 0 0 1 0 1;
                  0 0 0 0 1 0;
                  0 0 0 0 0 1');
Q = numpy.matrix('0.25 0 0 0 0.5 0;
                  0 0.25 0 0 0 0.5;
                  0 0 0.25 0 0.5 0;
                  0 0 0 0.25 0 0.5;
                  0.5 0 0.5 0 1 0;
                  0 0.5 0 0.5 0 1');
Q[:] = [a*0.01 for a in Q];
H = numpy.matrix('1 0 0 0 0 0;
                  0 1 0 0 0 0;
                  0 0 1 0 0 0;
                  0 0 0 1 0 0');
R = numpy.matlib.identity(4);
R[:] = [b*42.25 for b in R];
P = numpy.matlib.identity(6);
P[:] = [c*10000 for c in P];

n = 1;
while True:
    number = '%08d'%(n);
    filename = ('output/bbox_' + number + '.csv');
    if not os.path.isfile(filename):
        break;

    # read in the detected object's bounding box dimensions
    fid = open(filename);
    fid.readline(); #ignore the header
    detections = [];
    for i in fid:
        detections.append(i.strip().split(' '));
    fid.close();

    meas_x1 = float(detections[0][0]);
    meas_x2 = float(detections[2][0]);
    meas_y1 = float(detections[0][1]);
```

```
    meas_y2 = float(detections[2][1]);
    z = numpy.matrix([[meas_x1], [meas_x2], [meas_y1], [meas_y2]]);

    # step 1: predict
    result = kalman_predict(x,P,F,Q);
    x = result[0];
    P = result[1];

    # step 2: update (if measurement exists)
    if all(j>0 for j in z):
        result = kalman_update(x,P,z,H,R);
        x = result[0];
        P = result[1];

    est_z = H*x;
    est_x1 = est_z[0];}
```

[B] - CMT with Kalman Filtering and Video Streaming

```
#!/usr/bin/env python

import cv2
import argparse
import os.path
import time
import io
import subprocess
import CMT
import numpy
import util
import numpy.matlib
from numpy.linalg import inv

# Kalman Filter Setup

def kalman_predict(x,P,F,Q):
    x = F*x;         #predicted state
    P = F*P*F.transpose() + Q; #predicted estimate covariance
    res = [x, P];
    return res;

def kalman_update(x,P,z,H,R):
    y = z - H*x;     #measurement error/innovation
    S = H*P*H.transpose() + R; #measurement error/innovation covariance
    K = P*H.transpose()*inv(S); #optimal Kalman gain
    x = x + K*y; #updated state estimate
    P = (numpy.matlib.identity(x.shape[0]) - K*H)*P; #updated estimate

    res = [x, P];
    return res;
```

```
# Set up CMT parsing/necessary for writing csv data

parser = argparse.ArgumentParser(description='Track an object.')
parser.add_argument('--output-dir', dest='output', help='Specify a dire

args = parser.parse_args()

if args.output is not None:
    if not os.path.exists(args.output):
        os.mkdir(args.output)
    elif not os.path.isdir(args.output):
        raise Exception(args.output + ' exists, but is not a directory'

# define the filter

x = numpy.matlib.zeros((6,1));
F = numpy.matrix('1 0 0 0 1 0;
                  0 1 0 0 0 1;
                  0 0 1 0 1 0;
                  0 0 0 1 0 1;
                  0 0 0 0 1 0;
                  0 0 0 0 0 1');
Q = numpy.matrix('0.25 0 0 0 0.5 0;
                  0 0.25 0 0 0 0.5;
                  0 0 0.25 0 0.5 0;
                  0 0 0 0.25 0 0.5;
                  0.5 0 0.5 0 1 0;
                  0 0.5 0 0.5 0 1');
Q[:] = [a*0.01 for a in Q];
H = numpy.matrix('1 0 0 0 0 0;
                  0 1 0 0 0 0;
                  0 0 1 0 0 0;
                  0 0 0 1 0 0');
R = numpy.matlib.identity(4);
R[:] = [b*42.25 for b in R];
P = numpy.matlib.identity(6);
P[:] = [c*10000 for c in P];

n = 1;

CMT = CMT.CMT()
f =io.open('video.h264','rb')

if(True):

    counter = 0

    #Here we read frames from f and write them to a temp file
    red = f.read()
    w = io.open(str(counter)+'temp.h264','wb')
```

```
w.write(red)
w.close()#-loglevel panic

#This converts the h264 ti mp4
subprocess.call("ffmpeg -loglevel panic -y -i "+str(counter)+"temp.

# Is this necessary?
time.sleep(2)
print "here"
cap = cv2.VideoCapture("vid_files/im_00.mp4")
print "here"
status, im0 = cap.read()
im_gray0 = cv2.cvtColor(im0, cv2.COLOR_BGR2GRAY)
im_draw = numpy.copy(im0)

tl, br = util.get_rect(im_draw)

print "t1: " + str(tl) + " br: " + str(br)

CMT.initialise(im_gray0, tl, br)

frame = 1

#We always sleep between conversion and capture (do we need to?)
counter+=1
while True:
    # Read image
    status, im = cap.read()
    print "Status: " + str(status) + " of Frame #" + str(frame)
    if(counter%10!=1):
        counter+=1
        continue
    #If we've run out of frames for that mp4...
    if not status or counter == 1:
        cap.release()
        red= f.read() #Get more frames!!
        w = io.open(str(counter)+'temp.h264','wb')
        w.write(red)
        w.close()#-loglevel panic
        subprocess.call("ffmpeg -loglevel panic
        -y -i "+str(counter)+"temp.h264 im_0"+str(counter)+".mp4",
        # Is this necessary?
        time.sleep(1)
        cap = cv2.VideoCapture("im_0"+str(counter)+".mp4")
        counter+=1
        status, im = cap.read()

    im_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    im_draw = numpy.copy(im)
```

```
tic = time.time()
CMT.process_frame(im_gray)
toc = time.time()

# Display results
# Draw updated estimate
if CMT.has_result:

    cv2.line(im_draw, CMT.tl, CMT.tr, (255, 0, 0), 4)
    cv2.line(im_draw, CMT.tr, CMT.br, (255, 0, 0), 4)
    cv2.line(im_draw, CMT.br, CMT.bl, (255, 0, 0), 4)
    cv2.line(im_draw, CMT.bl, CMT.tl, (255, 0, 0), 4)

util.draw_keypoints(CMT.tracked_keypoints, im_draw, (255, 255,
# this is from simplescale
util.draw_keypoints(CMT.votes[:, :2], im_draw)  # blue
util.draw_keypoints(CMT.outliers[:, :2], im_draw, (0, 0, 255))

# Draw output image

cv2.imwrite('{0}/output_{1:08d}.png'.format
(args.output, frame), im_draw)

with open('{0}/bbox_{1:08d}.csv'.format(args.output,
frame), 'w') as z:
    z.write('x y \n')
    numpy.savetxt(z, numpy.array((CMT.tl,
    CMT.tr, CMT.br, CMT.bl, CMT.tl)), fmt='%.2f')

# Draw Kalman Filter here instead
number = '%08d'%(n);
filename = ('output/bbox_' + number + '.csv');
if not os.path.isfile(filename):
    break;

# read in the detected object's bounding box dimensions
fid = open(filename);
fid.readline(); #ignore the header
detections = [];
for i in fid:
    detections.append(i.strip().split(' '));
fid.close();

meas_x1 = float(detections[0][0]);
meas_x2 = float(detections[2][0]);
meas_y1 = float(detections[0][1]);
meas_y2 = float(detections[2][1]);
z = numpy.matrix([[meas_x1], [meas_x2],
[meas_y1], [meas_y2]]);
```

```
# step 1: predict
result = kalman_predict(x,P,F,Q);
x = result[0];
P = result[1];

# step 2: update (if measurement exists)
if all(j>0 for j in z):
    result = kalman_update(x,P,z,H,R);
    x = result[0];
    P = result[1];

est_z = H*x;
est_x1 = est_z[0];
est_x2 = est_z[1];
est_y1 = est_z[2];
est_y2 = est_z[3];
center = est_x1/2.0+est_x2/2.0
if(center>330):
    print "MOVE RIGHT"
    g = open('file.txt','w')
    g.write("1")
    g.close()
#if(center)
# draw a bounding box around the detected object
imgname = 'output/output_' + number + '.png';
img = cv2.imread(imgname, 1);

if all(k > 0 for k in est_z) and
(est_x2>est_x1) and (est_y2>est_y1):
    cv2.rectangle(img, (est_x1, est_y1),
    (est_x2, est_y2), (0,255,0), 2);

cv2.imshow('KalmanFilter',img);
cv2.waitKey(1);

n += 1;


#cv2.imshow('main', im_draw)

# Check key input
k = cv2.waitKey(1)
key = chr(k & 255)
if key == 'q':
    break

# Remember image
im_prev = im_gray

# Advance frame number
```

```
            frame += 1

    f.close()
```