Visual Human Detection and Tracking on a quadrotor platform.

Harsh Sinha (14265)
Dept. of Aerospace Engg.
harshsin@iitk.ac.in

Shubh Gupta(14670)
Dept. of Electrical Engg.
shubhgup@iitk.ac.in

Abstract—This report presents an overview of a Robot Operating System (ROS) based architecture for human tracking through a quadrotor using deep learning techniques for detection and Kalman Filter/Tracking techniques employed for generating smooth control input. Since quadrotors have low payload carrying capacity, we employ a remote server for the detection task. With this project we intend to develop a quad-copter based system for detecting and tracking a human target using single or multiple cameras with the ability to function appreciably in occluded and crowded environments, as well as being able to follow the target while maintaining a fixed separation as a surveillance task.

Index Terms—Quadcopter, Human tracking, ROS, Gazebo, human detection, SSD, KCF.

1 Introduction

Human-following robots have been researched and developed actively these decades due to their plentiful applications in daily life and manufacturing. There are already commercially available quadrotors which are being used by professionals for shooting videos of adventure sports. Although the current systems rely on the use of a mobile phone or some other beacon for correct tracking except for the one introduced by DJI with its Phantom 4 drone, which being a commercial product, is closed source.

A human-following robot requires several techniques such as human's target detection, robot control algorithm and obstacles avoidance. Human detection and tracking is a difficult task in general due to abrupt human object motion, object occlusion and object scale change, and changing object appearance due to changes in illumination and viewpoint, non-rigid deformations, intra-class variability in shape and posture, and potential camera movement, non-overlapping field of views between cameras. Furthermore, movement of camera with tilt and jerks accompanied by motion of the quadrotor make it difficult to achieve a fix over the position of a mobile target such as a human as the quad may lose sight of the target. Furthermore, state of the art techniques for detection and tracking use deep learning and require high computation powers and dedicated hardware to function at a reasonable rate.

With this project, we aim to create an open source system for accurate and real time system for human tracking which could perform well in crowded as well as slightly occluded conditions.

This project can be divided into the following prime sections : Detection of human subject, Tracking of the subject, Trajectory generation and Control of the vehicle. We discuss these in detail in the coming sections.

1.1 Previous Work

In recent years, the problem of human tracking using drones has received a lot of attention. Juhng-Perng Su et al. [1] and Tayyad Naseer et al. [2] use a stereo camera or a depth camera for multi-rotor drone tracking of a human by detecting a targeted object. Work by Thomas Muller and Markus Muller [3] uses monocular cameras to track a human who have different color against background colors. Work by Ashraf Qadir et al. [4] focuses on an unmanned miniature plane tracking an object by detecting the image similar to an image called template in two-dimensional images captured by a monocular camera. Another work by Imamura [5] tracks a human while detecting a human without the differences of colors and the movements of a target by making use of Histograms of Oriented Gradients (HOG) features and linear Support Vector Machine (SVM) for ROI classification.

In [6] Comaniciu et al. introduce a new framework for efficient tracking of nonrigid objects by spatially masking the target with an isotropic kernel. In [7] Weng et al. proposed an adaptive Kalman filtering algorithm to effectively track the moving object in a video frame sequence. Liu et al. and Redmon et al. have worked on methods using deep learning [8], [9] to detect a bounding box around an object in an image.

Achtelik et. al. in [13] work on a system where motion of a quadcopter is stably controlled based on visual feedback and measurements of inertial sensors. Bartak et. al. [14] utilize a computer-vision approach called tracking-learning-detection (TLD) to track an arbitrary object selected by a user in the videostream going from the front camera of the drone. Dang et. al. [15] perform a systematic formulation of a closed-loop control system design for tracking a ground object using ardrone platform. In the PhD thesis of Kalal [16] the authors propose a novel tracking framework (TLD) that explicitly decomposes the long-term tracking task into tracking, learning, and detection, where the tracker follows the object from frame to frame. In a seminal work by Lucas and Kanade, [17] the authors present a new image registration technique that uses spatial intensity gradient information to direct the search for the position that yields the best match in stereo image pairs.

2 Methodology

In this section we discuss the pipeline for the system which consists of the following submodules

- Environment Simulation
- Stereo image processing module
- Remote machine object detection node
- Distance estimation
- Kalman filtering
- Object of Interest Tracking
- Control node

We will discuss these submodules in more detail in upcoming sections.

2.1 Environment Simulation

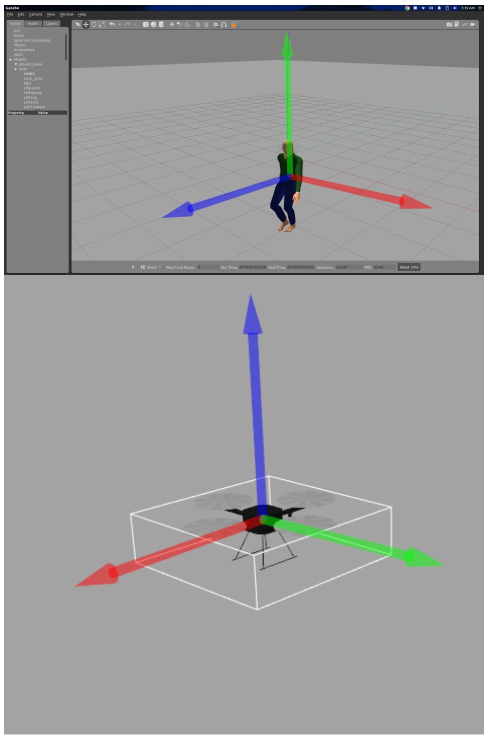
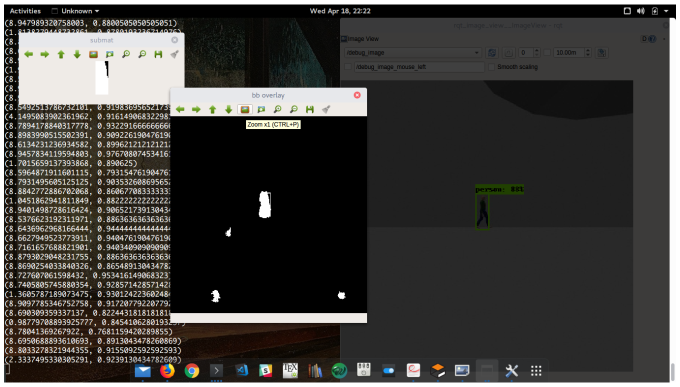


Fig. 1: Simulation models of quadrotor and human subject

Developing the pipeline requires a simplified testbed which simulates the real world to a certain extent, in order for the developed pipeline to be robust and be directly adaptable to real world use cases with minimum modifications. We have used Gazebo 8.0 for simulation purposes, since it has direct integration with Robot Operating System (ROS). A custom world was created for the purpose, consisting of a human model with waypoints set to walk in a square, and a quadcopter with stereo camera mounted on it. The images acquired by the quadcopter were published onto ROS topics so that they are accessible just like a real camera image. In order to test for occlusions we have also included a building. The simulation uses 'ros-control' package for realistic quadrotor controls, this package is also used on real world robot applications.



```
(8.947989320758003, 0.8800505050505051)
(1.8138279448732861, 0.8780193236714976)
(8.
(8.
(8.
(1.
(8.
(8.
(8.
(8.549251378673210l, 0.91983695652173
(4.1495083902361962, 0.916149068322981
(8.7894178840317778, 0.93229166666666
(8.8983990515502391, 0.90922619047619
(8.6134231236934582, 0.89962121212121
(8.9457834119594803, 0.97670807453416
(1.7015659137393868, 0.890625)
(8.5964871911601115, 0.79315476190476
(8.7931495605125125, 0.90353260869565
(8.8842772886702068, 0.86067708333333
(1.0451862941811849, 0.88222222222222
(8.9401498728616424, 0.90652173913043
(8.5376623192311971, 0.88636363636363
(8.6436962968166444, 0.94444444444444
(8.6627949523773911, 0.94047619047619
(8.7161657688821901, 0.94034090909090
(8.8793029048231755, 0.88636363636363
(8.8690254033840326, 0.86548913043478
(8.727607061598432, 0.95341614906832
(8.7405805745880354, 0.92857142857142
(1.3605787189073475, 0.93012422360248
(8.9097785346752758, 0.91720779220779
(8.690309359337137, 0.82244318181818
(0.98779708893925777, 0.845410628019323
(8.78041369267922, 0.768115942028985)
(8.6950688893610693, 0.891304347826086)
(8.8033278321944355, 0.915509259259259)
(2.3337495330305291, 0.923913043478260)
```

Fig. 2: Disparity map

2.2 Stereo image processing module

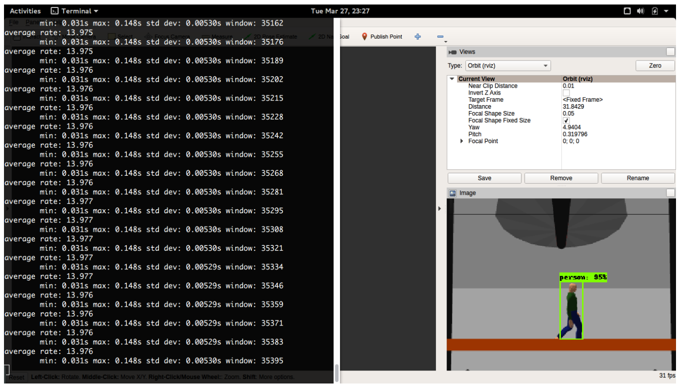
Stereo images acquired from the simulation were used to calculate a disparity map, by making use of relative positions of the camera. The specs of the simulated camera are shown in table 1.

The disparity of features between two stereo images are usually computed as a shift to the left of an image feature when viewed in the right image. In real world applications since stereo images may not always be correctly aligned to allow for quick disparity calculation, images obtained are first rectified for the relative rotations of the cameras. After rectification, the correspondence problem can be solved using an algorithm that scans both the left and right images for matching image features to compute disparity by the normalized correlation.

$$NC = \frac{\sum\sum L(r,c).R(r,c-d)}{\sqrt{(\sum\sum L(r,c)^2).(\sum\sum R(r,c-d)^2)}} \tag{1}$$

We make use of the "stereo_image_proc" package in ROS to calculate and publish the normalized correlation disparity.

2.3 Remote machine object detection node



```
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35162
average rate: 13.975
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35176
average rate: 13.975
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35189
average rate: 13.976
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35202
average rate: 13.976
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35215
average rate: 13.976
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35228
average rate: 13.976
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35242
average rate: 13.976
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35255
average rate: 13.976
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35268
average rate: 13.976
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35281
average rate: 13.977
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35295
average rate: 13.977
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35308
average rate: 13.977
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35321
average rate: 13.977
        min: 0.031s max: 0.148s std dev: 0.00529s window: 35334
average rate: 13.977
        min: 0.031s max: 0.148s std dev: 0.00529s window: 35346
average rate: 13.976
        min: 0.031s max: 0.148s std dev: 0.00529s window: 35359
average rate: 13.976
        min: 0.031s max: 0.148s std dev: 0.00529s window: 35371
average rate: 13.976
        min: 0.031s max: 0.148s std dev: 0.00529s window: 35383
average rate: 13.976
        min: 0.031s max: 0.148s std dev: 0.00530s window: 35395
```

Fig. 3: Object detection along with rate on remote server

Deep Learning based approaches trained on the large datasets like Image-Net, such as SSD (Single Shot multibox Detector) [8], faster R-CNN [12], YOLO (You Only Look Once) [9] etcetera have been consistently performing well

Quantity	Value
Image Resolution	400x400
h_fov	80 degrees
baseline	20 cm
Update Rate	30 Hz

Table 1. Stereo Camera Specifications

for the classification of different objects, including humans. Detection is done using these methods in order to identify humans in the camera image from the drones. A simple comparison of the speed of the different methods leads to the top contenders, YOLO and SSD. Now, one of the big issues with these approaches, stopping them from real-time application in drones is the need of Graphical Processing Units (GPUs). These GPUs are heavy and power-hungry, thus cannot be placed on drones, also the smaller Nvidia Jetson boards have quite under-powered ARM based CPUs limiting small application as well.

The solution that we propose is to use the Robot Operating System (ROS) in order to transfer the images captured to a separate computer on ground and then do the processing there and transfer the results back using the same architecture. This would allow us use huge computers with great computing power, which would then in turn allow us to use heavier and better performing techniques. This proposal will be limited by the bandwidth of the connection, as of now, this is being circumnavigated by sending images of smaller sizes (see table 1). In order to further increase the rates of transfer we are only sending compressed images, which provide huge boosts to the processing rates (see table 2 for detailed time analysis).

At present we are using the SSD, from the tensorflow object detection API [11] released sometime back.

2.4 Distance estimation

Using disparity image calculated from stereo image matching as well as the object bounding box obtained from the detection node, we estimate the distance of the object from the camera. From the bounding box region extracted from the disparity image, the largest cluster of similar-valued pixels is extracted and identified to be the cluster belonging to the human. A mean value of the pixels belonging to this cluster is calculated, which is used to obtain the depth estimate by the image projection formulae

$$z = \frac{f.T}{d} \tag{2}$$

where z is the distance to be estimated, d is the disparity value, f is the focal length of the camera in pixels and T is the baseline in real world units.

Another monocular camera based approach was developed for depth estimation in the project, using the scaling factors obtained from measuring the spread of ORB features in the image. The algorithm proposed was as follows

- Detect ORB features in the image
- Remove features lying outside bounding box for the image
- match the features in consecutive images and remove unmatched features
- calculate std. deviation of the features in x and y for both t and t+1 image frames
- obtain estimates of rate of change of depth using aspect ratio of bounding boxes
- remove the estimate (x or y) with greater change from previous value

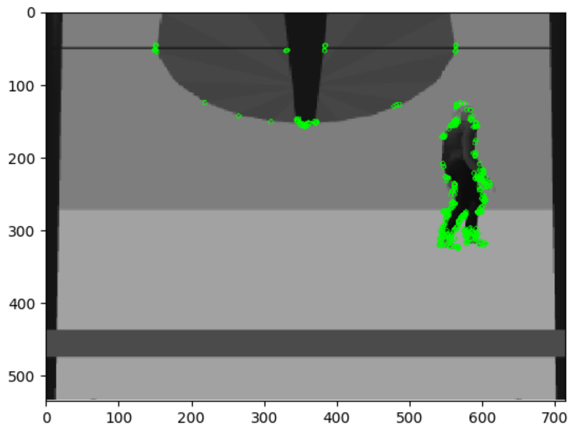


Fig. 4: ORB features in image frames

The algorithm makes use of the fact that change in depth of an object from camera is inversely proportional to its visible size in the image and directly proportional to the true size of the object. Hence,

$$\delta z = k.l(\frac{1}{x_t} - \frac{1}{x_{t+1}}) \tag{3}$$

where δz is the change in true depth of the object, x_t, x_{t+1} are sizes of projection in images at t and t+1 frames, l is the true size of object and k is proportionality constant. We estimate the size of the object in the image by the std deviation of the ORB features inside the bounding box in the image. Since this value is susceptible to changes by occlusion, we use the true size of the visible portion as the true size of the object in the formulae. Also adding the assumption the the object will be either occluded in x direction or y direction at a time,

$$\delta z_x = k.(w/L)(\frac{1}{std_x^t} - \frac{1}{std_x^{t+1}}) \tag{4}$$

$$\delta z_y = k.(l/W)(\frac{1}{std_y^t} - \frac{1}{std_y^{t+1}}) \tag{5}$$

where w,l are true sizes of visible portions and W,L are total true sizes of the objects. Hence, quantities $w/L, l/W$ can be estimated by aspect ratio and its inverse of the smaller bounding box in the images.

Finally, the stereo method was chosen since it gave more robust depth estimates.

Devices	Image Capture	Detection	Image Transfer Uncompressed	Compressed
Intel i5-3250, No-GPU	34 ms	130 ms	NA	NA
Nvidia Geforce 960M	34 ms	60 ms	NA	NA
Nvidia Titan X	34 ms	5 ms	500 ms	10 ms

Table 2. Time Analysis

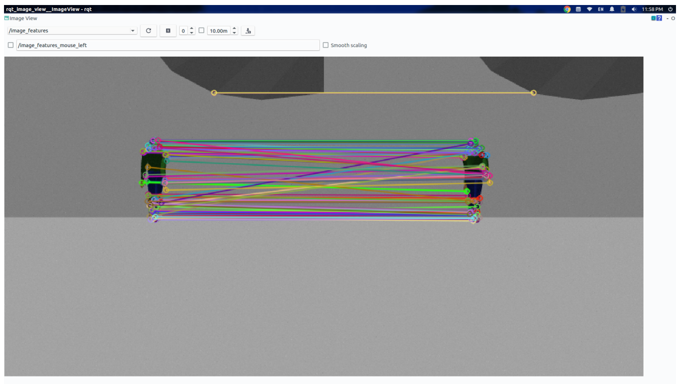


Fig. 5: Matched ORB features in consecutive frames

2.5 Kalman filtering

The estimates of bounding box coordinates obtained from object detection node, as well as the depth estimates obtained from stereo-disparity estimates and rate of change of depth estimation from ORB features were combined to be fed as measurements for the relative state of the object(human) in pixel coordinates into the kalman filter. The Kalman filter, running at 30 Hz, was employed to smooth out the obtained measurements as well as provide predictions in case the measurement generating nodes failed to provide any measurement. Furthermore, the predictive capability of the kalman filter was also leveraged to provide measurements in case the object to be tracked was occluded, since the measurements will not be generated in such cases as well.

A vanilla Kalman filter can be modelled as

$$\begin{aligned} x_0 &= Gaussian(\mu_0, \Sigma_0) \\ x_{t+1} &= A_t.x_t + b_t + \epsilon^1_{t+1} \\ y_t &= C_t x_t + d_t + \epsilon^2_t \\ \epsilon^1_t &= Gaussian(0, Q) \\ \epsilon^2_t &= Gaussian(0, R) \end{aligned} \tag{6}$$

where x is the model state and y are the measurements. For our case

$$x_t = \begin{bmatrix} x_c \\ y_c \\ z_c \\ v_x \\ v_y \\ v_z \\ a_x \\ a_y \\ a_z \end{bmatrix} \qquad y_t = \begin{bmatrix} x_m \\ y_m \\ z_m \\ v_z \end{bmatrix}$$

The state-transition and observation matrices for tracking human subject

$$A_t = \begin{bmatrix} 1 & 0 & 0 & T & 0 & 0 & 0.5*T^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 & 0 & 0.5*T^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & T & 0 & 0 & 0.5*T^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & T & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$C_t = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

with b_t and d_t set to 0.

Values of process and measurement noise parameters Q and R are determined from measurement set using Expectation-Maximization, which is a way to find maximum-likelihood estimates for model parameters when your data is incomplete, has missing data points, or has unobserved (hidden) latent variables. It is an iterative way to approximate the maximum likelihood function. It works by choosing random values for the missing data points, and using those guesses to estimate a second set of data. The new values are used to create a better guess for the first set, and the process continues until the algorithm converges on a fixed point. The EM algorithm:

$$\begin{aligned} Q(\theta|\theta^t) &= E_{Z|X,\theta^t}(logL(Z, X, \theta)) \\ \theta^{t+1} &= argmax_\theta Q(\theta|\theta^t) \end{aligned} \tag{7}$$

Where measurements are modelled as X and Z is the latent variable over which distributions are formed. The values of noise parameters (modelled by θ) are estimated during the 2nd step (M-step).

2.6 Object of Interest Tracking

The implementation of this project in the real world would certainly command the need to use wireless networks for information exchange between the ground based server and the quadrotor. Now, the prime options for communication of this kind would be 4G/5G mobile networks or WiFi, give the short range of WiFi, for long range applications one would gravitate towards the former option. One of the biggest issues which we anticipate this method would have to deal with, would be the occasional frame drop when transferring image data at high rate. In order to deal with this we use a ROS implementation of "High Speed Tracking with Kernelized Correlation Filter" [18]. Our implementation is based off of code from Tomas Vojir [19].

KCF kernel

$$k_{xx'} = exp(-\frac{1}{\sigma^2}(||x||^2 + ||x'||^2 - 2F^{-1}(\tilde{x}.\tilde{x'}))) \tag{8}$$

Where x and x′ are image patches being compared. This computation can be performed in O(nlog(n)).

The idea behind this method is that give one (or more) labels one should be able to perform online learning in order to discriminate between patches of interest and the background. More details on the original implementation can be easily found. Now since this is a very less computationally involved, the rate of performance goes as up as high as 100 frames per second. This lets us relax up the condition on the detection pipeline. As for the first labeled frame, we have used the detection bounding box returned after the execution of first frame.

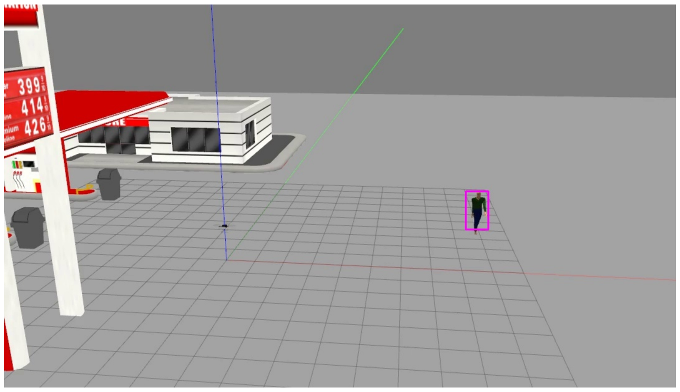


Fig. 6: KCF Tracked human subject

2.7 Control node

The rectified control input is obtained from Kalman filter/ Tracking node. The input is fed into a hierarchical controller, highest level of which uses a cascade of two PID controllers for the horizontal movement and two separate PID controllers for the yaw rate and vertical velocity. The velocity outputs are used for feeding into the lower level controller which calculates the necessary forces and torques acting on the body. The torques and forces hence generated are then applied on the simulation to make the quadrotor move. The low level control is is handled by Gazebo plugins, and hence we tune the high level controller.

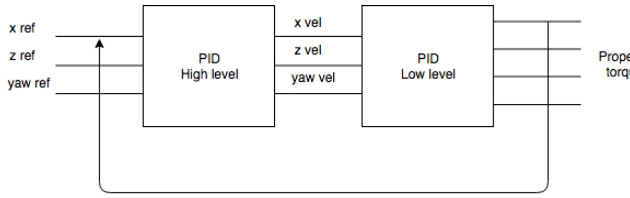


Fig. 7: Cascaded controller overview

3 Experiments

3.1 Till Midterm report

A gazebo simulation is used for all of the experiments , the simulation used was originally created by Ritesh Haldhar and modified by us. We are using a simulation with a quadrotor and several *actor* based humans moving on certain trajectories, which at the moment are fixed. The quadrotor used is *hector-quadrotor* and for capturing images the onboard camera is used, the resolution of capture is 320x240. Now, the reason we can claim that methods discussed before, which are based on feature point detection like [7] can work as when testing for detection of ORB features on the obtained image we got ¿400 feature points.

Images captured are sent to GPU server on which a tensorflow API is used to perform detection using SSD, this is done to simulate the real world implementation of the pipeline. The frequency of the camera was set to 15Hz as the fastest detection was being done at 14Hz and feeding in more frames would only consume bandwidth of the connection used to transfer images between the GPU server and simulation.

3.2 After Midterm report

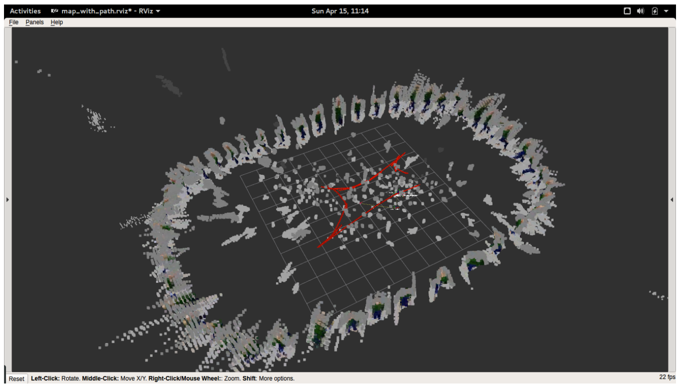

Activities
map_with_path.rviz* - RViz
Sun Apr 15, 11:14
File Panels Help
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
22 fps

Fig. 8: Tracectory of quadrotor with human subject point cloud

Support was added for more complex trajectories of the human subject such as moving in a square and with varying speeds and turn rates. This allows the simulation to better model real world scenario. Also, this increased a challenge for quadrotor since the subject no longer permanently stayed inside the camera field of view. This also helped avoid local optima states for the quadrotor where it could keep the subject at the center by just changing the yaw values.

Depth estimation portions were worked upon both in using stereo camera as well as monocular version, since the previous method of estimating the distance by the area of the bounding box was unreliable. The area based estimate changed quadratically with distance, voiding the linear requirement of the simple control, as well as changed drastically in case of occlusion or if the human subject was at an edge of the frame.

Kalman filter was implemented for high rate control input generation as well as occlusion avoidance. This helped take care of the issue of missing detections in hard-to-detect frames as well as smoothed out the detections, aiding in easier control. It also provided an added functionality of ability to incorporate occlusion/ out-of-frame cases.

Tracking node was developed for object detection in crowded environments. KCF was incorporated into the pipeline along with kalman filter to provide better estimates since it also uses information directly from the frames unlike kalman filter. This provided an added functionality of ability to incorporate cases with multiple moving subjects.

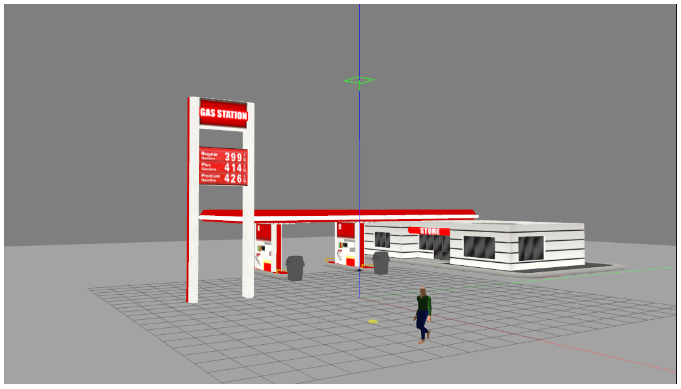


Fig. 9: Experimental setup in Gazebo

4 Results

Detailed time based analysis can be obtained in Table 2 for various submodules in the pipeline.

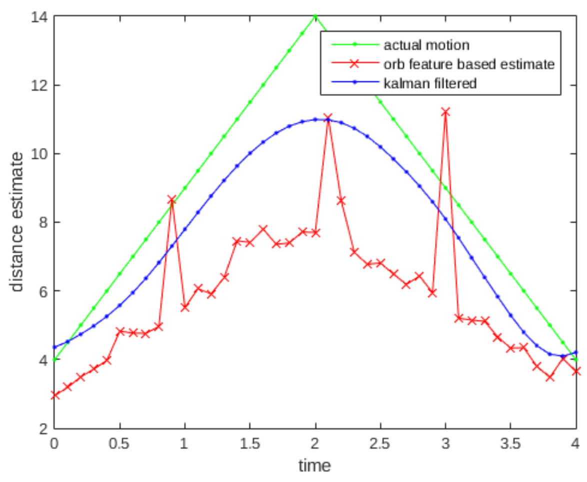


Fig. 10: Results of distance estimation by Kalman filter as well as orb feature method against ground truth

5 Conclusion

We developed a system for detecting and following a human subject autonomously via a quadrotor with only vision based techniques. The payload over the quadcopter was kept to a minimum by offloading high computation tasks to a remote server with a GPU. The system was made robust to occlusion and/or presence of other subjects via KCF tracking and Kalman filtering. The proposed system was tested to be able to function in a Gazebo simulation.

Limitations of the proposed system include the requirement of a high speed and bandwidth connection to the remote server machine to transport images and similar data. Hence, functioning over WiFi hence results in suboptimal results. Another limitation is that the initial frame seen by the quadcopter should only contain the target human subject, since the system cannot decide between different subjects in the first frame. Also, the system is not yet robust to heavy jerks in the camera video feed, as well as fails when it loses the sight of the human target for longer periods of time.

6 Future Works

"One of the virtues of working on simulation is the lack of real world problems one has to face irrespective of how accurately the simulation models the real world, since the real world is just broken." Besides the implementation on a real world quadrotor, the following things can be worked on as extensions to this project:

- Exploration of trajectory generation techniques and maybe a move towards optimal trajectory-generation.
- The model can be edited to include a gimbaled camera which would reduce the adverse effect on the camera image due to the motion of the quad.
- Other filters from the kalman family can be exploed.

References

[1] Juhng-Perng Su and Kuo-Hsien Hsia, Height Estimation and Image Tracking Control of an Indoor Quad-Rotor Craft via Multi-Vision Systems, International Journal of Computer, Consumer and Cool (IJ3C), vol. 2, no. 4, 2013

[2] Tayyad Naseer, Jurgen Sturm and Daniel Cremers, FollowMe: Person Following and Gesture Recognition with a Quadrocopter, 2013IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS), November, 3-7, 2013

[3] Thomas M ller and Markus M ller Vision-based drone flight control and crowd or riot analysis with efficient color histogram based tracking, SPIE 8020, Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications VIII, 80200R, May, 25, 2011

[4] Ashraf Qadir, Jeremiah Neubert, and William Semke, On-Board Visual Tracking with Unmanned Aircraft System (UAS), Infotech Aerospace 2011, March, 29-31, 2011

[5] Imamura, Yusuke, Shingo Okamoto, and Jae Hoon Lee. "Human tracking by a multi-rotor drone using HOG features and linear SVM on images captured by a monocular camera." Proceedings of the International MultiConference of Engineers and Computer Scientists. Vol. 1. 2016.

[6] Comaniciu, Dorin, Visvanathan Ramesh, and Peter Meer. "Kernel-based object tracking." IEEE Transactions on pattern analysis and machine intelligence 25.5 (2003): 564-577.

[7] Weng, Shiuh-Ku, Chung-Ming Kuo, and Shu-Kang Tu. "Video object tracking using adaptive Kalman filter." Journal of Visual Communication and Image Representation 17.6 (2006): 1190-1208.

[8] Liu, Wei, et al. "Ssd: Single shot multibox detector." European conference on computer vision. Springer, Cham, 2016.

[9] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[10] Caelles, Sergi, et al. "One-shot video object segmentation." CVPR 2017. IEEE, 2017.

[11] Tensorflow "Tensorflow Object-detection API" https://github.com/tensorflow/models/tree/master/research/object_detection 2017.

[12] Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems. 2015.

[13] Achtelik, Markus, et al. "Visual tracking and control of a quadcopter using a stereo camera system and inertial sensors." Mechatronics and automation, 2009. icma 2009. international conference on. IEEE, 2009.

[14] Bartk, Roman, and Adam Vykovsk. "Any object tracking and following by a flying drone." Artificial Intelligence (MICAI), 2015 Fourteenth Mexican International Conference on. IEEE, 2015.

[15] Dang, Chi-Tinh, et al. ”Vision based ground object tracking using AR. Drone quadrotor.” Control, Automation and Information Sciences (ICCAIS), 2013 International Conference on. IEEE, 2013.

[16] Kalal, Zdenek, Krystian Mikolajczyk, and Jiri Matas. ”Tracking-learning-detection.” IEEE transactions on pattern analysis and machine intelligence 34.7 (2012): 1409-1422.

[17] Lucas, Bruce D., and Takeo Kanade. ”An iterative image registration technique with an application to stereo vision.” (1981): 674-679.

[18] Henriques, Joo F., et al. ”High-speed tracking with kernelized correlation filters.” IEEE Transactions on Pattern Analysis and Machine Intelligence 37.3 (2015): 583-596.

[19] Vojir, Tomas, ”Open Source implementation of High-speed tracking with kernelized correlation filters.” Github:https://github.com/vojirt/kcf