Major Project - DroneX

MTRX5700 - Experimental Robotics

470355499

470355503

***Abstract*—This paper outlines the design and implementation of a low cost proof-of-concept autonomous drone to be used as a production aid for on-stage videography. The aim of this system is for it to be able to reliably track a 'presenter' on-stage whilst they move around presenting. It was also desired that the presenter be able to use the system alone, therefore requiring the system to receive some form of user input via the camera. The developed system, hereon dubbed 'DroneX', utilises a low cost off-the-shelf drone to perform all videography, in addition to a typical GPU enabled laptop for the computation required to control the drone for steady video output. Machine learning is used in conjunction with PID controllers to command the drone to track a main 'presenter' within the frame. The prototype for DroneX as discussed in this paper was able to successfully achieve the product goal, using a Parrot AR Drone 2.0 to robustly following an inframe user as they moved about slowly. A natural user interface was also implemented through OpenPose to allow users to manipulate the drone via simple arm-based gestures.**

Introduction

When filming on stages such as at TED Talks, where the presenter freely moves about onstage, one or more camera operators are usually required. As seen in Figure 1, the equipment these cameramen carry are often bulky and take up a lot of space onstage, blocking the scene for members of the audience and negatively impacting their experience. Current solutions to this problem have typically been focused on reducing the size of this equipment, however the problem still persists as these systems remain ground based and manned.

Fig. 1: Stage Camera from 'A Star is Born'

This paper proposes using an automated airborne drone as a filming device to minimise onstage occlusions and reduce the number of required personnel. The implementation of this technology would have significant impact on the media industry in reducing labour costs and improving audience experience. This automation could also be used as a commercial product for individuals to film themselves for personal projects. As such, an emphasis is placed on producing a versatile system that enables a low cost drone to perform complex tasks by leveraging other existing hardware and technologies. DroneX should be able to autonomously track the presenter, who will be moving around during the presentation. The presenter should also be able to control the drone through simple gesture commands, to manipulate the composition of the camera frame. Developing this technology, particular the people tracking component, can also be beneficial to other fields involving human-robot interactions such as semi-automated factories with helper robots who follow human operators.

The aim of this paper is to explore whether a low cost drone is able to be used as an autonomous airborne solution for on-stage videography through external computing. Relevant previous work will be investigated first, followed by details on the design and implementation of the system. The results will then be presented and discussed, along with significant challenged encountered. Lastly, future directions of the project will be outlined for a more accurate and efficient second generation.

Background

Drone photography had long been the interest of many, mainly due to the relatively unrestricted positioning of the camera on the drone. The obvious questions raised from this idea is drone control - how can drones be controlled so that good photo shots can be taken?

If the drones were to operate autonomously, an additional question will be raised - how can drones identify and track the object of interest?

These two questions, regarding drone control and object tracking, need to be addressed with extra care when designing an autonomous video recording drone system.

Research has been done into controlling a quadrocopter drone in the past few years. However, these systems are designed to either simply track an object of interest through computer vision, or bases all commanding off external outputs such as keyboard and human pose[1][2][3]. There is no system available currently

that performs object tracking as well as human gesture commanding at the same time.

Drones are inherently difficult to control due to their highly dynamic nature, which makes tracking an object of interest continuously also difficult[4]. Accurate drone control can be achieved by utilising multiple carefully tuned proportional integral derivative (PID) controllers for each rotational axis, based on kinematics of rotors, sensors and sensor models[4].

Luckily, the commercial product Parrot AR-Drone has pre-developed drone controllers allowing the users to control and access flight data through the SDK. A Robot Operating System (ROS) package based off the Software Development Kit (SDK) has been designed for Parrot AR-Drone in 2015[5]. The Parrot AR-Drone package, along with middleware ROS developed by Stanford University in 2007 greatly reduced the effort required to interface with the quadrocopter hardware to perform basic drone maneuvers and extract flight data[6].

Similar work involving object tracking on Parrot AR-Drone had been done in 2016 by Chakrabarty et al. [1]. The project utilised ROS to interface with the drone hardware, and implemented Consensus-based Matching and Tracking of Keypoints for Object Tracking(CMT) to track the object of interest, and Image Based Visual Servoing (IBVS) to control the drone autonomously and follow a moving 3D object[1]. The CMT performs object tracking through extracting features and key points from current and past images. Good results of rigid and deformable object tracking were demonstrated in real world test flights[1]. However, this system simply tracks the the object of interest in frame, and is unable to identify what exactly it is tracking. Additionally, the system takes no external user input, and hence it is impossible for the user in frame to adjust drone tracking settings. Whilst not explicitly stated, it is assumed that Chakrabarty et al. used an external device for computation given that they are using the same hardware as we intend.

Facial detection and tracking had also implemented on drones. X. Sun and W. Zhang had demonstrated facial detecting utilising Haar-like feature evaluation and AdaBoost algorithm[2]. Simple user commands such as takeoff, land, move up/down/left/right can also be given to the drone through keyboard[2]. Although this system can be controlled by the user, the device is relatively large, not very portable, and uses an unnatural user interface.

A commercial video filming product was released in 2018 by Skydio. The original system, Skydio R1, as well as the second generation, Skydio 2, both utilised more than 10 on board cameras and neural networks to track a fast moving person of interest whilst performing obstacle avoidance[7]. On board GPS was also used to help with drone localisation[7]. However, the impressive capability of the system also demands high onboard computational power to operate neural networks on large amount of data. The hardware requirements brings the both systems to a relatively high price, $2500 and $999 USD for Skydio R1 and Skydio 2 respectively, which is not desirable for most customers[7]. The system is encapsulated and requires no external device for computing but takes inputs from the user through external devices, such as a smart phone or a controller.

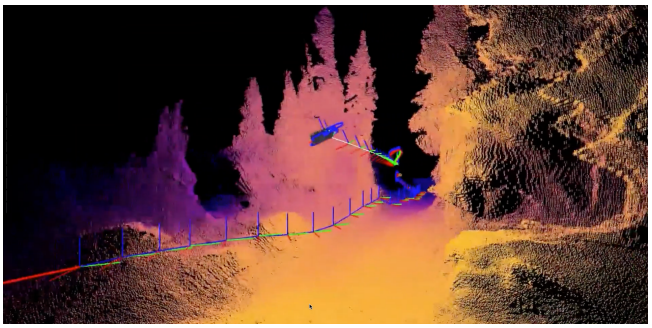


Fig. 2: Skydio Tracking Object of Interest

Projects had also explored the possibility of utilising Natural User Interface (NUI) to control aerial vehicles. A paper published in 2020 investigated utilising NUI to command a drone through a user friendly and intuitive way[3]. In this paper, body gestures was extracted by OpenPose, a real time multi-person human pose detection library, to send a variety of commands to manoeuvre the drone[3]. The system consisted of the drone, Parrot Bebop 2, and a laptop acting as a ground station, with ROS, OpenPose, CUDA, CUDNN and caffe installed[3]. The system focused on making controlling the drone comfortable and intuitive, and was tested on people with various prior experience with drones[3]. Whilst the gesture control was successful, no implementation of person tracking was performed, the commands were simply applied to the drone based on its current pose.

Experimental Setup

The DroneX experimental setup is composed of three key subsystems; the Drone Controller, the Target Tracker and the Gesture Detection system. These three components work in tandem to compute appropriate command velocities for controlling the motion of the drone. Here we first present the overall system architecture, followed by discussion of the detailed design and implementations of the key subsystems.

Overall System Architecture

DroneX is built atop the low-cost Parrot AR Drone 2.0 which conveniently features all required hardware for the system; an RGB camera, IMU and a hackable flight controller.

Designed originally to be controlled by a downloadable phone app, ROS is instead used to facilitate communication between the AR Drone and an external computer for performs off-board computing. The ARDrone Autonomy ROS package is used to interface all of the drone's functionality with the device, allowing image and odometry data to be published over ROS topics. Conversely, the package also allows command velocities to be published to the drone and dictate its movements.

The laptop used for external processing has an Intel Core processor i7-8750H, Nvidia GeForce GTX1050 and 16 GB of RAM. The operating system used was Ubuntu 16.04, with ROS Indigo, CUDA 8.0, CuDNN 6.0 installed. Additionally, all DroneX software modules were build as ROS nodes with Python 2.

The final key technology used in DroneX is OpenPose, which was used to extract human pose data from an image. A ROS wrapper is used to integrate OpenPose into our ROS pipeline and directly obtain images from the drone for inference.

This system architecture is summarised in Figure 3.

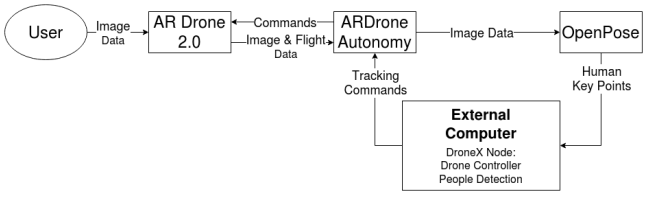


Fig. 3: DroneX System Architecture

Drone Controller

To control the movements of the drone in order to track a target, a controller was required. The ability of the drone to track a visible target is limited by the quality of the actuators and the control system of the drone. As the aim is to test the ability of a low-cost drone system, this was expected to a degree. Instead, a focus was placed on refining the controller for the AR Drone. That said, the controller was not complex and classical PID controllers were implemented. The PID controllers control the output, or the command velocity of the drone by taking into account the proportional, integral and differential of the error in the drone's position. The PID controller is shown below, where $u(t)$ is the outputted command velocity, $e(t)$ is the error and K_i, K_p, K_d are integral, proportional and derivative gains respectively.

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_p \frac{de}{dt}$$

Each of the three gains have different effects on the behaviour of the system. Increasing the proportional gain K_p will increase the speed the system approaches its reference, however will also increase overshoot. K_i behaves similarly to K_p, however also reduces the steady state error of the system to zero. Increasing the derivative gain K_d would decrease the overshoot of the system.

Three different PID controllers were implemented to independently control linear motion in the X,Y and Z axes, as well as an additional one over yaw for rotational motion. To enable rapid prototyping, a green vision target (detergent box) was used as a placeholder for the drone's Target Finder module output. This allowed all the controllers to be tuned quickly without relying on any other modules.

This controller was utilised for two forms of tracking, relative and absolute. These two would be used exclusively in different circumstances, with the prior for pure pursuit tracking of a presenter in frame and the latter for direct tracking of a pose in the world.

Relative Tracking: For relative tracking, the visible camera scene was searched for a viable 'target' by the Target Finder. This target was then compared with a predefined reference, and the errors in the target size and position were used for feedback control. A sample of this can be seen in Figure 4.

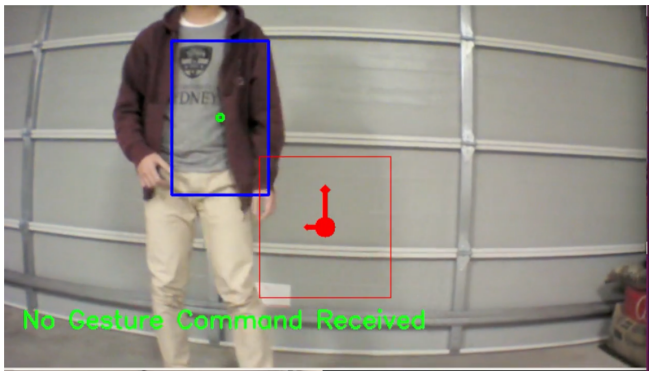


Fig. 4: Relative Tracking

For motion in the z-y plane, the positional errors between the target and reference centres in frame were directly used by the control loop. In the x direction however, such linear offsets were not possible due to the lack of depth data. Instead, the heights of the target and reference were compared for percentage error, which was then fed into the x-axis controller. The height was chosen over the width of the body, as the torso height extracted by OpenPose is generally more consistent compared to the width.

$$x_{err} = (1 - \frac{h_{target}}{h_{ref}}) * 100$$
$$y_{err} = y_{ref} - y_{target}$$
$$z_{err} = z_{ref} - z_{target}$$

Visual feedback on the controller's behaviour is provided in the form of vectors onscreen, also seen in Figure 4. If the target is successfully tracked (errors within acceptable bounds) or if the target is lost in frame, the innate hover functionality of the AR Drone is used to maintain the current position until the error grows or the target is found again.

Relative Tracking should behave identically to the tests with the green vision target, thus should not need further independent testing.

Absolute Tracking: Absolute tracking was also implemented to allow the DroneX system to track its own position in the world and be commanded to specific goal poses. This would enable more advanced features such as waypoint following, returning to base and maintaining steady poses. Additionally, the drone could also be designed to track not only the relative location of the presenter, but also map their pose, allowing the drone to manoeuvre itself to ensure the presenter is always facing forward towards the camera.

Although the Ardrone Autonomy ROS package predefines an 'odom' frame, it was found to seemingly randomly centre the yaw. It was decided that it made sense for the drone to be commanded with respect to where it first took off. Thus, the takeoff pose was recorded and used as a transformation to centre all future odometry readings. This centred frame was defined as the new 'world' frame of the drone.

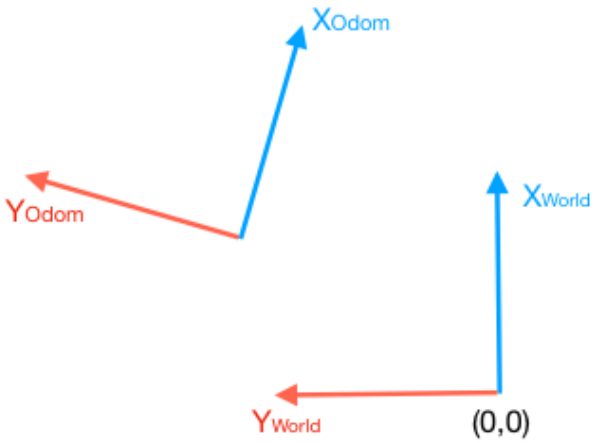


Fig. 5: World Frame and Odom Frame

Initially, the takeoff pose was taken directly as the starting pose of the drone whilst still on the ground. However, due to some abstracted feature of the ARDrone Autonomy package, the pose was perpetually zeroed before takeoff, and after takeoff, would sometimes converge to a seemingly random x and y coordinate. This often did not match the physical drifting of the drone during takeoff. Hence, a delay was required before sampling the starting pose to account for these errors.

After investigating the raw odometry plots via bagfiles, shown in Figure 6, it was found that the odometry's x-y drift correlated quite closely with the drone's initial increase in altitude. Therefore, it was chosen that the first decrease in flight data altitude would mark the end of the takeoff phase. This allowed the system to bypass the x-y takeoff drifts and produced a relatively stable/accurate world frame relative to the takeoff pose for absolute pose control.

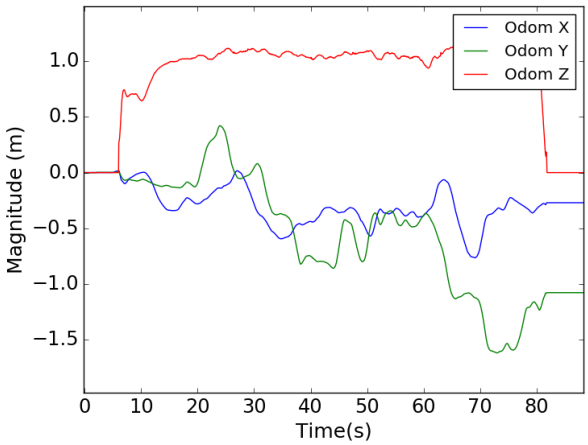


Fig. 6: Drone Odometry

It was also found that the altitude measurement of the drone was usually accurate, likely because it uses both the IMU and the downwards facing ultrasonic sensor for altitude determination. Because of this, the world frame's zeroed altitude could still be set at approximately the 'true' ground height, which helped lower the risk of collision with the ground.

With the world frame established, the system can continually track the drone's current position using flight odometry. For actually controlling the drone motion, the linear errors in x, y, z and yaw(α) are given as inputs to the Drone Controller. In order for error correction to be correctly performed, the errors found need to be transformed from the world frame to the current (relative) robot frame, before the drone actuates relative to its current pose. The equations for this transformation are shown below:

$$
\begin{aligned}
x_{err,world} &= x_{goal} - x_{target} \\
y_{err,world} &= y_{goal} - y_{target} \\
x_{err,rel} &= x_{err,world}\cos(\alpha) - y_{err,world}\sin(\alpha) \\
\delta y_{err,rel} &= x_{err,world}\sin(\alpha) + y_{err,world}\cos(\alpha) \\
z_{err_r el} &= z_{goal} - z_{target}
\end{aligned}
$$

To test this, goal poses were published to the system using a custom ROS message and observing the resultant error plots and onscreen vectors. The physical drone would also need to be observed for expected behaviour - the drone should fly to approximately the correct goal pose relative to where it ended after takeoff.

Target Finder

The Target Finder enables the AR Drone to identify a visible target to follow. OpenPose is a real-time human key point detection system[8] utilising neural networks to identify up to 135 key points of the human body[8]. An example output is shown in Figure **??**, where each detected body part has a fixed index.

As mentioned previously, DroneX leverages an OpenPose ROS wrapper to directly feed images from the drone through the network to produce human pose data. To ensure robust people tracking and to reduce computer hardware requirements, the target tracker identifies people using only the key points of the torso. By using these larger and more prominent points a lower OpenPose 'net resolution' was needed (meaning inference was less computationally intensive) and allowed for an improvement in FPS of around 10. To offset the lower accuracy due to lower net resolution, checks were implemented to ensure that the found key points are infact corresponding to humans in the scene. These checks include:

- Human torso aspect ratio should be roughly 1:1 width to height with height always greater than width.
- Torso must have left and right shoulder and hip points.
- Torso must have minimum height on the screen.

Whenever a valid torso is found in frame, a bounding box is drawn around it, which then becomes the found 'target' used by the control system. At present, if multiple valid torsos are found, then only the main (largest) torso is tracked as a target.

To test this subsystem, the AR Drone was propped up to chest height to simulate an in-flight video feed. A user could then walk around in frame and the outputted tracking data could be overlaid on the video to confirm robust identification of people.

Gesture Detection

The key point data from OpenPose is also used to enable Gesture Detection from the user in frame. By taking unique combinations of upper body limb positions, different commands could be sent to the drone such as:

- Hover the drone - Stop target tracking
- Unhover the drone - Resume target tracking
- Pan left - Move the drone further left of the tracked target
- Pan right - Move the drone further right of the tracked target
- Land - Land the drone

For Gesture Detection, we move beyond just using the torso key points and instead use the upper body limbs, consisting of the wrist, elbow and shoulder points. These points were chosen as the drone only focuses on the torso for tracking, hence would only reliably see the upper body. For similar reasons, all gestures were contained within the upper half of the body to ensure any actions made by the user would be consistently captured.

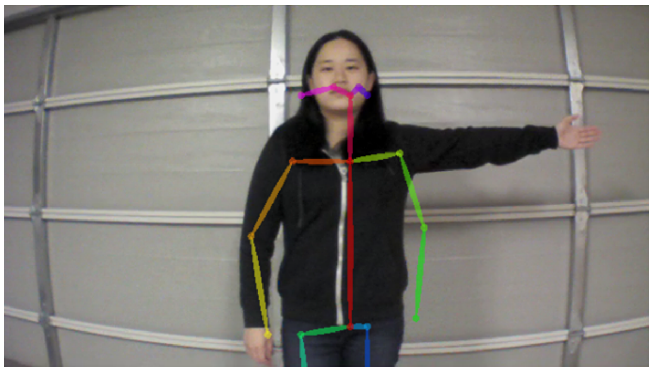


Fig. 7: OpenPose Arm Out Failure Case

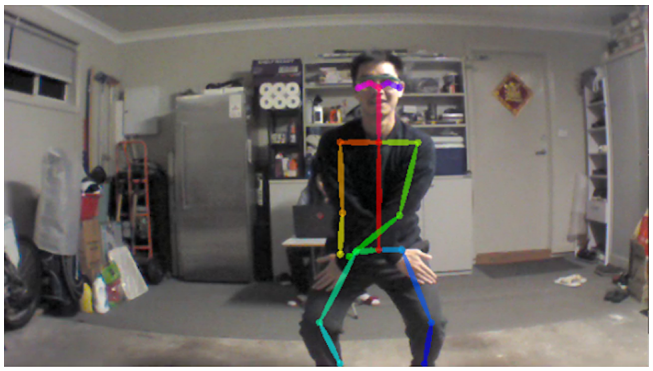


Fig. 8: OpenPose Arms Crossed Failure Case

It was found through experimentation that the module threw minimal false positives, but OpenPose did have some states where it under-performed. These included when the user:

- Had their arms straight out, parallel to the ground, as seen in Figure 7.
  We suspect this was due to low training data for this human pose.
- Had any overlapping limbs/joints such as their arms crossed. This can be seen in Figure 8.
  This was expected as the occlusion can confuse the model into not differentiating left and right limbs.
- Was partially out of frame.
  Predictably, the model did not reliably identify arms when notable features such as the hands were not in frame.

Hence, all gestures were uniquely identified by their relative positions to the torso and the obtuse angles made at the elbow key points. Some other changes included,

- Widening the tolerances for the Pan left/right gestures to $\pm 17^o$ error from horizontal.
- Making the Land gesture to hands together at the centre of the hip rather than arms crossed over at the hip.

The final list of gestures implemented are as follows:

- Hover the drone - Both arms out with $\pm 17^o$ tolerance from the horizontal.
- Unhover the drone - Both elbows straight out with hands over collarbone.
- Pan left - Left arm straight out with $\pm 17^o$ tolerance from the horizontal.
- Pan right - Right arm straight out with $\pm 17^o$ tolerance from the horizontal.
- Land - Both elbows out with hands on head.

These permissible gestures were specifically chosen to not interfere with a presenter's natural body language whilst presenting, however still be natural enough for them to use without being too obvious. Examples of this include panning left or right, which requires completely straight and outstretched arms. Typically, people loosely move their limbs, rarely locking them, hence this gesture will not interfere with the presenter but can still be utilised subtly.

The Gesture Detection submodule also functions independently from the Target Finder and across gestures. If for example the left wrist could not be found, any gestures involving the left arm would be impossible, however right arm based commands such as 'Pan right' would still be able to be performed. Additionally, the existence or non-existence of any of these wrist/elbow points has no impact on the Target Finder as it only

cares about the torso.

To test Gesture Detection, the same setup as the Target Tracker testing was used. A user would stand in frame of the camera and perform various movements to determine if the algorithm was robust.

Results

Reflecting DroneX's modular system design, experimentation and testing was also performed in a modular manner before integration. Because of reasons soon to be discussed, collecting quantitative data to present in this paper was challenging. Thus, to best showcase the results that were achieved, two videos are provided here: https://youtu.be/HYobsUteGCA, https://youtu.be/wEBmqmAks5U.

Firstly, some general issues with the drone used were encountered during testing.
The drone would often behave unreliably, changing in behaviour throughout the day. At the beginning of each day, the drone would behave ideally, with minimal drift. However, by the afternoon, it would often takeoff, then begin spuriously rotating at a rate of about one revolution a minute.
Another issue encountered was the drone sometimes failing to directly takeoff. After sending a takeoff command from the external computer, sometimes the drone would fail to vertically takeoff and do a 'hop' of sorts forward before getting to height. However, testing was generally not possible when this issue occurred. This did not happen too often but would usually be resolved by a reboot of the entire system.
The odometry of the drone also had some major issues, which will be discussed in further detail shortly.

Additionally, the computational ability of the computer used was found to have a non-trivial impact on the behaviour of the system. In general, it was found through replaying bagfiles on a higher specced desktop that the frame rate of the OpenPose output was considerably lower on the laptop. Because the Relative Tracking was reliant on this outputted frame rate, the controller loop rate was also lowered. The accuracy of the human key point poses was also diminished on the laptop as the desktop could use a higher net resolution whilst still achieving higher frame rates.

Drone Controller

It was found that the quality of the actuators of the AR Drone used was actually fairly low. This in conjunction with the relatively low computational capacity of the laptop used meant that the controller responsiveness was slow and had a tendency to be erratic when dealing with overshoot. Hence, the PID gains were tuned to minimise overshoot and instead favor a smooth approach towards the target.

Tuning the Drone Controller was relatively straight forward once it was decided to use different gains for each of x, y, z and yaw motion. For the three linear axes, once tuned, the drone was seen to be smoothly tracking the provided green vision target with a settling time of around six seconds. In the end, our linear motion PID controllers had a K_i of zero, essentially reducing them to PD controllers. The addition of this extra complexity provided no visible benefit to the system's controllability and instead, upon investigation, was found to quickly saturate despite very small gains - causing excessive overshoot as the controller took time to unwind the integral error.

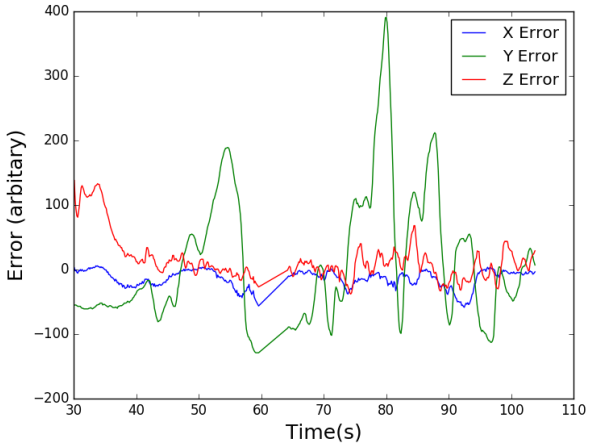


Fig. 9: Target Tracking Errors

Figure 9 shows the error plots of the drone whilst tracking a moving target. Unfortunately, due to the lack of accurate odometry, the error in the system cannot be expressed in the world frame, and instead only relative error can be visualised in plots such as the one above. Here, the x and z axes were relatively consistent, thus the system is able to control the drone to maintain the desired target. In the y-axis however, the user was constantly in motion, thus the drone was constantly chasing them to try minimise the error. In the produced

videos and in person, the drone could be clearly seen robustly tracking the target in 3-dimensions.

As just mentioned, it was found that the drone's odometry was rather inaccurate, having a tendency to fail significantly, producing garbage results that were completely different to what was observable in reality. Because of this, the drone's pose was unable to be tracked and as a consequence, the yaw controller could not be properly implemented. Figure 10 demonstrates the odometry drift in yaw after the drone performed a 360^o rotational drift. Clearly the measurements do not correspond with reality and similar results were seen in the linear axes also.

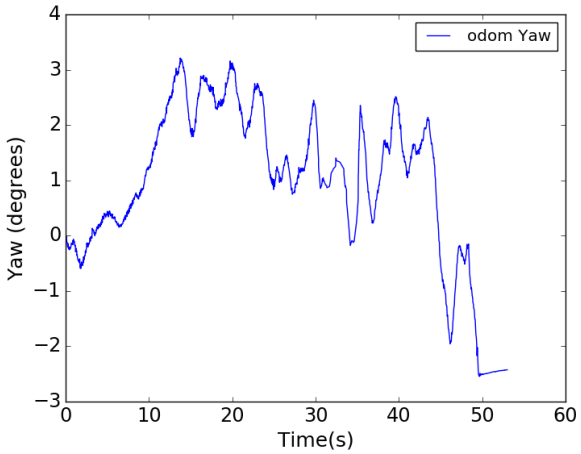


Fig. 10: Odom Drift

Because of this, further testing for the absolute tracking was not possible as the drone localisation was extremely inaccurate.

Target Finder

The reliability of this submodule and the next was dependent on the performance of the OpenPose ROS wrapper on the laptop, which was adequate. This was judged based on the accuracy of the outputted key points and frame rate of the wrapper's output. With a moderate net resolution of 656x320, a frame rate of around 15 FPS and generally consistent key point detection for the two authors under typical indoor lighting conditions was obtained.

OpenPose was able to consistently identify the key points of the torso for individuals in frame, even when they were in motion or side-on to the camera. Because of this, the Target Finder was able to continuously track a user in the visible camera frame. The implemented sanity checks were also seen functioning as expected, rejecting missing persons with strange aspect ratios or missing key points. Figure 11 showcases some examples of this robustness in the module.

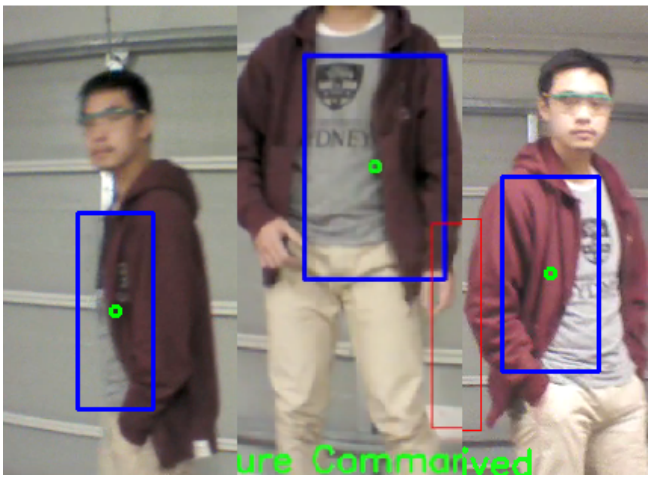


Fig. 11: OpenPose detecting a partially occluded user

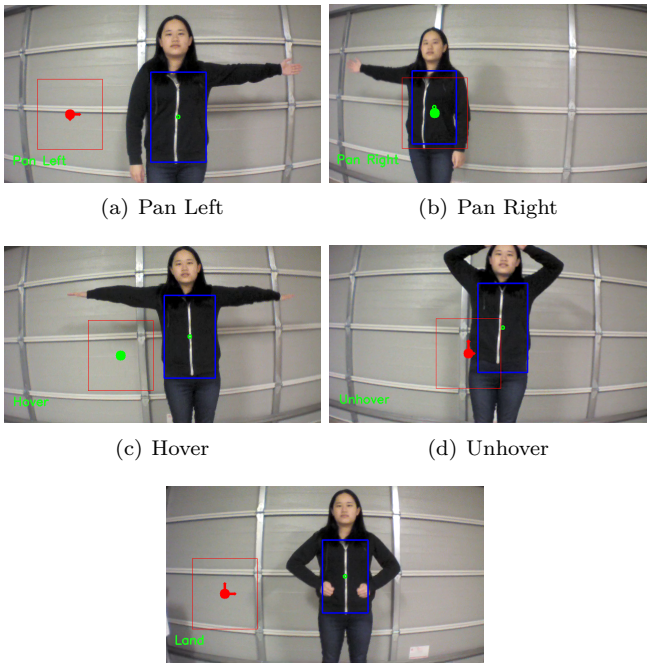
However, there were still some caveats in the system where tracking broke down. Notably, reliability significantly decreased as distinguishing features such as the head or hips exited the camera frame. This occurred when the user got too close to the drone and caused OpenPose to return insufficient key points. The opposite action also provided lower reliability, as the user getting too far away from the drone meant body parts were less defined and harder for OpenPose to classify.

Gesture Detection

The performance of the Gesture Detection module was evaluated based on how responsive and accurate the software was at detecting gestures made by the user. With OpenPose's aforementioned general reliability, Gesture Detection worked well in isolation. Users were able to intuitively send basic commands to the drone with their arms given minimal direction on how to do so. On the laptop, the module was seen to respond quickly by shifting the reference target as soon as the gesture was performed.

The loosened restrictions on the panning and hovering gestures greatly improved their robustness and made the user gesture interface feel more natural to use. Successful detection of each gesture is presented in Figure 12. However, the 'deadzone' (Figure 7 when the user had their arms straight out and parallel to the ground

was still prevalent and caused some discontinuities in the module behaviour for related panning and hovering gestures.



(a) Pan Left (b) Pan Right

(c) Hover (d) Unhover

(e) Land

Fig. 12: DroneX Gestures

Integrated System Performance

With each of the core subsystems validated in isolation, the full DroneX system was brought together and tested within a typical double garage. The system performed well with tracking people consistently in a cluttered scene whilst they moved at a slow to moderate pace. The tracking was robust in all three linear axes and was even able to follow a main user whilst other persons were present in the background. As designed, the drone moved at a slow pace, roughly one tenth of its maximum speed. The video footage recorded by the camera also showed very smooth tracking of the user as they walked around.

Users were also able to 'present' naturally, with no major changes to body language required to accommodate the gesture commands. At the same time, these presenters were able to incorporate the gesture commands relatively seamlessly into their body language to subtly control the drone. Previously it was found that the gestures performed by the users registered quickly in software. In practice however, the slow rise time of the Drone Controller meant the physical drone was delayed in responding, thus sometimes the user would often overshoot panning the drone.

Different challenges were thrown at the system in the form of erratic and fast movements, testing its ability to track a very dynamic target whilst not picking up any false positive gestures.

Some failure cases that were found for the system include:

- If the target ducked sometimes they would exit the frame and no longer be tracked.
- If the target moved too far or too close OpenPose would break down and tracking would stop.
- If the target asked the drone to pan too far to the left or right sometimes their far-side arm would not be fully in frame, preventing re-centring of the drone.

Discussion

The results of DroneX's preliminary design and testing proved promising and successfully achieved the initial project goals to an acceptable level. During testing however, many observations were made that are worth further discussing in terms of causes, consequences and possible improvements.

While the general issues with the drone were likely not a manufacturing error, rather a consequence of excessive use and age (given that other similar projects don't have these issues[1]), they did provide us with many challenges and impacted our ability to test the system. As the drone seemed to work fine at the beginning of each day, it was reasoned that possibly the system needed significant cooldown time between each run. Unfortunately, this meant that testing of the system could only really be done once an hour roughly, although this did produce much more consistent results.

Another general issue with the low cost drone was noisy actuators. While no quantitative data was collected, being physically present while the drone was in flight proved very noisy. The level of noise produced was unacceptable for presentation contexts, however it is believed that this can be significantly reduced with smoother bearings and a propeller redesign.

As previously mentioned, we were unable to rely on the drone's internal IMU/odometry to localise itself. Because of this, the Absolute Tracking system had to

be completely benched due to time constraints for the project. As such, the current system moves exclusively via Relative Tracking and features such as returning to 'base' or maintaining a steady heading 'onstage' were completely out of the question. This was disappointing as they would have been practical additions to the system. Possible ways of improving this could involve getting a newer set of hardware or implementing a form of robot localisation such as SLAM. This will be further discussed in Future Work.

Again, due to the lack of accurate odometry data as well as the abstraction of using the Drone Autonomy package, it was difficult to produce a model of the drone dynamics to tune the controllers with. Therefore, it was decided that systematically tuning each gain through experimentation would be most practical given the circumstances. This was relatively straight forward by tuning each axis in isolation.

The computational ability of the laptop used proved to be quite restricting also, as it lowered the frame rate of the OpenPose wrapper significantly. Because the controller loop rate was effectively diminished when run on the laptop, it can be concluded that given better hardware for the external computer, the responsiveness of the Drone Controller has a lot of room for improvement. Additionally, there is a possibility that the previous requirement of no overshoot would no longer be needed. This was originally enforced because it was believed that the unresponsiveness of the Drone Control was due to actuator inadequacy, rather than slow computation rates as it is now suspected. We believe that this is an acceptable circumstance as the computational hardware available to media producers will typically be highly specced and a low cost drone will perform well.

The 'net-resolution' parameter of OpenPose was also decreased in order to maintain a usable frame rate on the laptop, about 14FPS. Whilst this reduction in resolution significantly increased the frame rate, it also reduced the accuracy of OpenPose detection, as shown in Figure 7. Overall, this lower accuracy was tolerable and could be accounted for through some basic sanity checks. However, the prevalent 'deadzone' was a bit off-putting at times for users as finding the 'sweet spot' could be finicky. Ideally, this deadzone be eliminated entirely. Again, replaying a recorded bagfile on a desktop showed lower a frequency of this error occurring. Thus, if the external computer used had better computational power, the OpenPose net-resolution could also be increased to ensure a more robust detection of persons, and hence a more reliable system for gesture detection.

Future Work

In order to further develop the DroneX system, the existing hardware limitations will firstly need to be overcome. If the hardware of the system could be improved, it would open up a range of possible extensions to the system. That said, the goal of this project is to test a low cost drone's potential, thus in the future, best performance may come from designing our own drone system. The benefits of improving the hardware is outlined below nonetheless.

With better actuators, the drone will be able to better maintain its pose and achieve more concise manoeuvres. Additionally, future motors and propellers should be chosen with a focus on quiet performance to minimise background noise for the presentation, taking inspiration from the latest DJI Mavic models[9]. Better accuracy and preciseness in the sensors would allow the system to localise the drone better. This would allow features such as the absolute drone controller to be used, which would in turn enable features such as waypoint setting and automatic landing at user defined locations.

Implementing some form of robot localisation would also be a key future development to make better use of the onboard sensors, opting for quantity over quality of sensors. Sensor fusion algorithms such as an Extended Kalman Filter could be implemented to improve system performance in such a way.

Improving the effective computational rate of the external computer would also help the system to make more dynamic and accurate decisions. Given that the system currently uses packaged software such as OpenPose, the easiest way to improve computational performance would be to improve the hardware - better GPUs and CPUs. This would increase the OpenPose detection frame rate as well as allow a higher resolution to be used for human detection, allowing the neural network to produce accurate results faster. Faster detection and data processing will also improve the responsiveness of the controller, leading to a smoother, better controlled system. Furthermore, the OpenPose library could extract more key points of the human body, allowing a variety of more subtle gestures to be used to interface

with the drone.

Another way to improve this would be to better optimise the code, refactoring it to a more optimised language such as C++. This would provide a general performance increase over all devices used, and should be done regardless. Further alternatives to improving general computational ability would be to embed the Drone Controller directly into the flight controller whilst leaving the rest of the computation to be performed externally.

Tracking algorithms could also be implemented such that the drone would track a specific person, instead of just largest detected person in frame. This would be useful in more complicated environments with multiple people in frame at close proximity. An example usage would be having the drone actively follow the host on crowded stages. This could be further extended to pass the centre of focus from person to person, through special gesture controls.

Conclusion

This paper presented the successful design and implementation of a autonomous drone system, DroneX, that could be used for onstage videography. Through external computing, a low cost off-the-shelf commercial drone was shown to be perfectly capable of performing complex people tracking. The results presented showed the system to be able to reliably track a slow human pace within a small stage, as well as the ability to recognise and act upon gesture commands robustly without false positives during a typical presentation. There is still plenty of room to improve DroneX through having better on-board actuators and sensors, more off-board computational power, using more optimised code and implementing tracking algorithms. Overall, the system was a successful proof-of-concept that a low-cost, easily accessible drone could be used for onstage videography. Therefore, we believe there is great potential for this technology and it should be further developed for possible commercial rollout in the future.

References

[1] A. Chakrabarty, R. Morris, X. Bouyssounouse, and R. Hunt, "Autonomous indoor object tracking with the parrot ar.drone," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2016, pp. 25–30.

[2] X. Sun and W. Zhang, "Implementation of target tracking system based on small drone," in *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 1, 2019, pp. 1863–1866.

[3] B. Yam-Viramontes and D. Mercado-Ravell, *Implementation of a natural user interface to command a drone*, 2020. arXiv: 2003.02662 [cs.HC].

[4] W. Yang, M. Chun, G. Jang, J. Baek, and S. Kim, "A study on smart drone using quadcopter and object tracking techniques," in *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, 2017, pp. 1–5.

[5] M. Monajjemi. (2016). Ardrone_autonomy, [Online]. Available: https://ardrone-autonomy.readthedocs.io/en/latest/.

[6] ROS. (2020). History, [Online]. Available: https://www.ros.org/history/.

[7] Skydio. (2020). Skydio 2, [Online]. Available: https://www.skydio.com.

[8] T. Digumarti. (2019). Mtrx 5700 : Experimental robotics - robot learning, [Online]. Available: https://canvas.sydney.edu.au/courses/22155/pages/robot-learning?module_item_id=721168.

[9] D. S. Central. (2020). 4 effective ways to make a quadcopter quiet, [Online]. Available: https://soundproofcentral.com/keep-quadcopter-quiet/.