Team 1:

Indoor UAV Tracking System

Final Project Documentation

Supervisor: Prof. Costas Armenakis
Industry Advisor: P.Eng. Lui Tai

May 01, 2018

Kevin Arindaeng (213094016)
Ariel Laboriante (212951984)
Zhuolin (Jack) Lu (212848834)
Varsha Ragavendran (213193065)

Source Code:
https://github.com/azkevin/U-TRACKR

Website:
https://azkevin.github.io/U-TRACKR/

1.0 Executive Summary

The U-TRACKR system provides low latency tracking for unmanned aerial vehicles (UAVs) in an indoor setting. The key objective is to locate the exact position of the UAV using only the image sequences of a limited four-camera system. Ultimately, the system will track and model the trajectory of autonomous UAVs using image-processing and photogrammetric techniques. The U-TRACKR software currently accomplishes time synchronization between the cameras, space intersection, space resection, and position calculations to obtain the X,Y,Z coordinates of the UAV. The system consists of four inexpensive Raspberry Pi camera modules fixed at the corners of a stable metal frame. A personal laptop with a high processing rate produces low latency calculations, manages the synchronization between the four Raspberry Pi camera modules, and runs the OpenCV software for object detection with minimum delay.

The advantages of the U-TRACKR system are threefold: First, it provides tracking solution for objects in indoor spaces where GPS is unreliable. Second the system is not hindered by frequency disturbances, temperature variances, and limited range as it would if the system were to use infrared sensors. Third, the U-TRACKR software is portable and any streaming camera can be used for the system. The initial design concept was targeted towards UAVs which are underutilized in indoor applications. The U-TRACKR will provide positioning technology which will improve efficiency and cost-effectiveness of programmable UAVs for indoor use. The system can be used in warehouses where UAVs have the advantages of a flexible flight path, automation, and the ability to enter environments that are dangerous to human life. In this context, the U-TRACKR system will fulfill the need for employee safety, increased productivity, and reduced company expenses allocated to damaged equipment. The U-TRACKR successfully provides a solution for robot trajectory control in indoor, GPS-denied spaces.

The scope of application can also be extended to tracking animals in captivity. Currently, zoos and researchers attach acoustic or GPS tags on animals to study their patterns. These techniques are invasive and unreliable in areas with GPS limitations. The execution of this project minimizes human interaction and disturbances, and provides a non-invasive solution for the study of animal lifestyle and behavior. The U-TRACKR can also be used to compare the popularity of indoor exhibits and stores. Some museums and art galleries are presently using IoT (Indoor Location) tracking with the help of visitor heat-maps to determine and compare the popularity of exhibitions. The U-TRACKR can provide real-time data collection with accurate human behavior trends by extending the software to use a machine learning library called TensorFlow. Further evolution of the U-TRACKR system in the future can provide complete robot or UAV path control for full automation.

Table of Contents

2.0 Document Overview

This is a technical blueprint for the U-TRACKR project. This document has been developed by Kevin Arindaeng, Ariel Laboriante, Zhuolin Lu, and Varsha Ragavendran for the capstone course, ENG4000: Engineering Project. This document is intended to satisfy all the requirements, objectives and expectations from the project *Statement of Work* and the *Final Year Project* governing document for project deliverables.

2.1 Document Conventions

The following acronyms listed in *Table 1* appear throughout this document.

Table 1: List of acronyms mentioned in the document

UAV	unmanned aerial vehicle	CPU	central processing unit
GPS	global positioning system	LSA	least squares adjustment
SSH	Secure Shell	HSV	lue-saturation-value
IoT	Internet of Things	CSA	Canadian Standards Associations
TCP	Transmission Control Protocol	3D	three-dimensional
FPS	frames per second	PVC	Polyvinyl chloride
Mbps	megabits per second	GCP	Ground control point
USB	universal serial bus	NOOBS	New Out Of the Box software
PC	personal computer	OTG	on-the-go

3.0 Project Requirements

3.1 Project Scope

The core of this project is to develop a tracking system that provides low latency navigation for objects placed within a defined area such as a room or warehouse. The main objective is to locate the position of the object in an indoor setting, where GPS is unreliable, using image-processing and photogrammetry.

The system consists of four Raspberry Pi modules with camera module placed at each corners of the cubed frame.

Originally, the project scope detailed the system to have a server setup that would allow the Raspberry Pi camera modules to stream the real-time video to the server with OpenCV software. Unfortunately, this design causes lag, as the Raspberry Pi modules have slower processing rate, thus, a real-time calculation is not possible.

Ultimately, the project scope expanded to incorporate the use of a main controller (laptop), which has a better processing rate and allows the system to produce low latency (1.0s to 3.0s) calculations. The low latency results are produced by managing the time synchronization for all four Raspberry Pi's and camera modules, while running OpenCV software with minimum delay.

3.2 Stakeholders

The main stakeholders of this project are identified below.

1. Businesses using robots within a warehouse

Businesses such as Amazon use mobile robots to improve their Prime services of promising two days delivery. However, these industrial robots such as Kiva, require a specific environment to operate. For example, floors need to have custom grids so that the robots can move, and workers are at risk of getting hit if they are in the way. The execution of this project will allow for businesses such as Amazon to track these mobile robots, and minimize safety-related lawsuit risks.

2. Scientists and researchers focused on animals in captivity

Presently, zoos are using techniques such as attaching acoustic tags or GPS tags on animals to carry out their research and capture movements, and behavioral patterns. These techniques are sometimes invasive and are not available for indoor areas where GPS is limited. The execution of this project minimizes human interaction and disturbances, along with providing a minimally invasive approach for tracking and studying the animal lifestyle and behavior.

3. Museums and art galleries comparing the popularity of exhibitions

Some museums and art galleries are presently using IoT/Indoor Location tracking with the use of visitor heat-maps to determine and compare the popularity of exhibitions. The execution of this project provides low latency data collection with more accuracy of behavior trends, as the project can incorporate the use of a machine learning libraries.

3.3 Engineering Requirements

3.3.1 User Requirements

The user will interface with the python command line and environment by executing the main.py program found within the U-TRACKR system, which will operate as expected within a minute. However, in order to successfully execute the program, the user will have to have the following libraries and frameworks installed:

Table 2: User requirements

Library/Framework to install	Command to Execute	Purpose
Python 2.7.13	install Python 2.7.13	The U-TRACKR system is entirely programmed in Python, therefore in order to execute this system the user will have to have the Python library installed.
OpenCV 3.4.0	install OpenCV 3.4.0	The OpenCV software is run through the tracker.py program which identifies the object through image processing techniques.
NumPy	pip install numpy	The NumPy module is used to define the HSV array values when for object tracking purposes using OpenCV.
SymPy	pip install sympy	SymPy is used for position calculation purposes, specifically in the space resection and intersection algorithm using its matrix libraries.
imutils	pip install imutils	This framework provides helper functions that go hand in hand with image processing operations such as translation, rotation, and resizing.
picamera[array]	pip install picamera[array]	This module is used to construct an n-dimensional array that captures the outputs from the cameras, which will then be used for image processing techniques.
paramiko	pip install paramiko	The paramiko framework allows the main controller to establish an SSH connection with each Raspberry Pi module to retrieve the live video feed.

3.3.2 System Requirements

3.3.2.1 Functional Requirements

Initially, the strategy was to incorporate the use of a server to process the video captured by the Raspberry Pi camera to identify the objects within the tracking area. However, it was realized that reading and writing to the server caused lag in the system. This defeated the purpose of identifying the trajectory of objects in real-time. Therefore, the system was redesigned to first establish an SSH connection with all four Raspberry Pi Zeros. A command is executed on each Raspberry Pi Zero to start and write the live video feed to the main controller (laptop) at the same time.

The U-TRACKR program was then successfully able to run the OpenCV software on the live feed to identify objects within the defined area. The U-TRACKR system then executes a Python script which calculates the coordinates of the object within the frame's area and outputs these coordinates to the console successfully.

Additionally, in the preliminary and critical design review documents, the functional requirements initially revolved around building a system that handles and detects only one object within the frame. This requirement was successfully met with the use of the OpenCV software that allows detection of objects based on color or ArUCO markers within a defined area.

3.3.2.2 Performance Requirements

The primary performance requirements have not changed with one minor exception. In the preliminary design review, the idea behind performing image processing was to first setup a server from which a Raspberry Pi module will process the videos frame by frame, to identify the objects in motion and identify the location of the objects in the indoor space. During implementation, major time delays were noticed when following this approach, thereby affecting the performance of the system. Hence, the current approach incorporates the use of a main controller instead of a server, which allows the system to produce real-time position calculation with minimum delay.

3.3.3 Regulatory Requirements

All regulatory requirements stated earlier in the preliminary design document were consistently met during the implementation of this project and the system complied with additional requirements. Since the system applies to indoor UAV applications in private warehouse buildings, there are no flying regulations or noise level requirements. The system complies with the Workplace Safety and Insurance Board (WSIB) policies, the Worker Health and Safety - Ontario Ministry of Labour policies, and the York University's policy with regards to Temporary Use of University Space. Firstly, as this project involves processing image frames from a live feed camera, privacy related regulations must be in effect when

operating this system with humans being in the defined area. Secondly, as the power interface uses commercial power supplies, it satisfies the CSA standards.

3.4 Limitations and Constraints

3.4.1 Limitations

The limitations of the U-TRACKR system are identified below:

Static Objects: Currently, the U-TRACKR system does not account for any static objects (non-moving objects) that may be within the frame. Although this may not be of big concern right now, when expanding this system to a bigger area, static objects must be taken into account as they may block the view of the cameras and therefore perform inaccurate calculations when determining an object's coordinates or the system might not be able to identify objects within the frame.

Identification of Multiple Objects: Although multiple objects may be placed within the frame, the system presently only tracks one specified object (Color, ArUCO trackers). While expanding to a bigger area, where multiple objects are to be tracked, the system must incorporate the use of machine learning algorithms which would account for multiple objects within the frame.

3.4.2 Constraints

The constraints of the U-TRACKR system are stated below:

WiFi requirement: Currently, the system establishes an SSH connection between the main controller and the four Raspberry Pi's, therefore WiFi connection is mandatory for system functionality.

Lux threshold: The U-TRACKR system is able to identify the frame of reference of an object within a 3D system provided there is ample amount of light available within the frame. If no light enters the frame, the U-TRACKR system will not be able to identify objects, and may produce inaccurate X, Y, Z coordinates.

3.4.3 Improvements

Possible improvements to the U-TRACKR system that would make it more suitable for actual use by stakeholders include:

Incorporating the use of Machine Learning Platforms: With the use of a machine learning platform such as TensorFlow, the U-TRACKR system can be evolve to learn and identify objects of various kinds within a defined area. This would allow the system to be more open to tracking objects of different kinds in various settings without having to manually program the system to track these kinds of objects.

Distinguishing Between Multiple Objects of the Same Type: The system did incorporate the use of a machine learning library to allow detection of various objects, however there were issues with distinguishing between multiple objects of the same kind. With this improvement in the future, the system would be able to track more objects within the defined area with no restrictions or limitations.

Better cameras with better processing rate: Presently, the system incorporates the use of Raspberry Pi cameras which have a slower processing rate and therefore affects the frames processed per second which in turn obstructs the objective to produce real-time position calculation. With the use of better cameras, the system would be able to determine the frame of reference of the objects at a faster rate and with a more precise calculation.

4.0 Engineering Concepts and Design Methods

4.1 Object Detection

In order for the U-TRACKR system to obtain a position calculation of an object, it first needs to detect and locate where an object is image-by-image using various computer vision techniques.

4.1.1 Image Acquisition

The U-TRACKR system performs image acquisition by streaming a video feed from each camera to a TCP connection. This is done remotely from the main controller by establishing an SSH connection to each Raspberry Pi, and running the following command:

```
'raspivid -t 0 -n -w 640 -h 480 -fps 30 -rot 90 -ex fixedfps -ex
auto -b 25000000 -o - | nc -l 5000'
```

This command utilizes the raspivid command line tool to produce a video stream at 480p resolution, with a fixed frame rate of 30 frames per second, an image rotation of 90 degrees, automatic exposure, and a bitrate of 25 Megabits per second. This command also utilizes the netcat utility to write the video data to the local TCP port of 5000.

The main controller then reads the video data by establishing a TCP connection to the Raspberry Pi using its IP address. It then uses the OpenCV VideoCapture class to obtain each frame from the video stream. Analysis of each image frame can then be done to perform object detection.

4.1.2 Detection by Color Space

Detection by color space is the extraction of distinct colours in the image given by the camera. The object must be distinguishable and have different color features than its surrounding.

OpenCV's findContours() function is used to detect a specific HSV color space range within the image frame. A specific lower and upper range is given for a particular object in order to detect it. Once the contour is obtained, a circle is drawn on the frame, and its centroid pixel coordinate is then used for position calculation.

In this case, the object being tracked is a tennis ball. Thus, an HSV range from (25, 70, 6) to (64, 255, 255) is used to find the contour below.

Figure 1. Tracking using HSV color space

One of the limitations of this detection method is that any other sources of green colour produced by the surrounding can affect the detection of the marker. Thus, the use of yellow light is also not recommended as yellow light interferes with the selected marker colour.

Another limitation is that if the lighting is too dark or too bright, the HSV range must be adjusted as well in order for the object to be detected in the image.

Figure 2. Limitations of detecting in low-light

4.1.3 Detection by ArUCO Markers

ArUCO markers are generally used in the determination of post-estimation in computer vision applications. These markers are composed of a black border and a binary pattern which determines its ID (identification). The black border allows for fast detection in the image frame.

OpenCV's ArUCO detectMarkers() function is used to detect a specific ArUCO marker in the image frame. Once it finds the black border and the correct binary pattern, a square is drawn around the object, and its midpoint pixel coordinate is then used for position calculation. In this case, this type of detection is used to track a drone. A 6x6 bit ArUCO marker is added on the top of the drone in order to detect it.

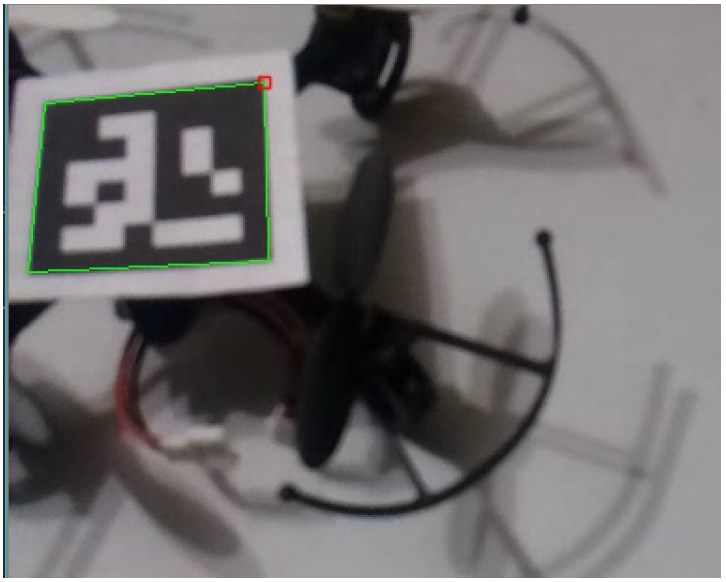


Figure 3. Detection of Drone using ArUCO Markers

One of the limitations of this type of tracking is that if the environment lighting is too dark, the system will not be able to detect the marker. Also, having the marker on any object becomes intrusive to it. In this example, having the marker on a drone limits its flying capabilities as a load is attached to it.

4.2 Time Synchronization

One of the main design processes for the U-TRACKR system to work is to get time synchronization working across multiple cameras. Since the position calculation of an object in 3D space is time-dependent, it is necessary to ensure that every frame captured by the camera happens at the same timestamp.

This is done using Python's threading interface to concurrently output each camera's timestamp, and the object's pixel coordinates relative to the frame at the same time. Each camera video stream starts at the same time, and after each sampling time of one second, the position calculation algorithm is performed based on each camera's pixel coordinates.

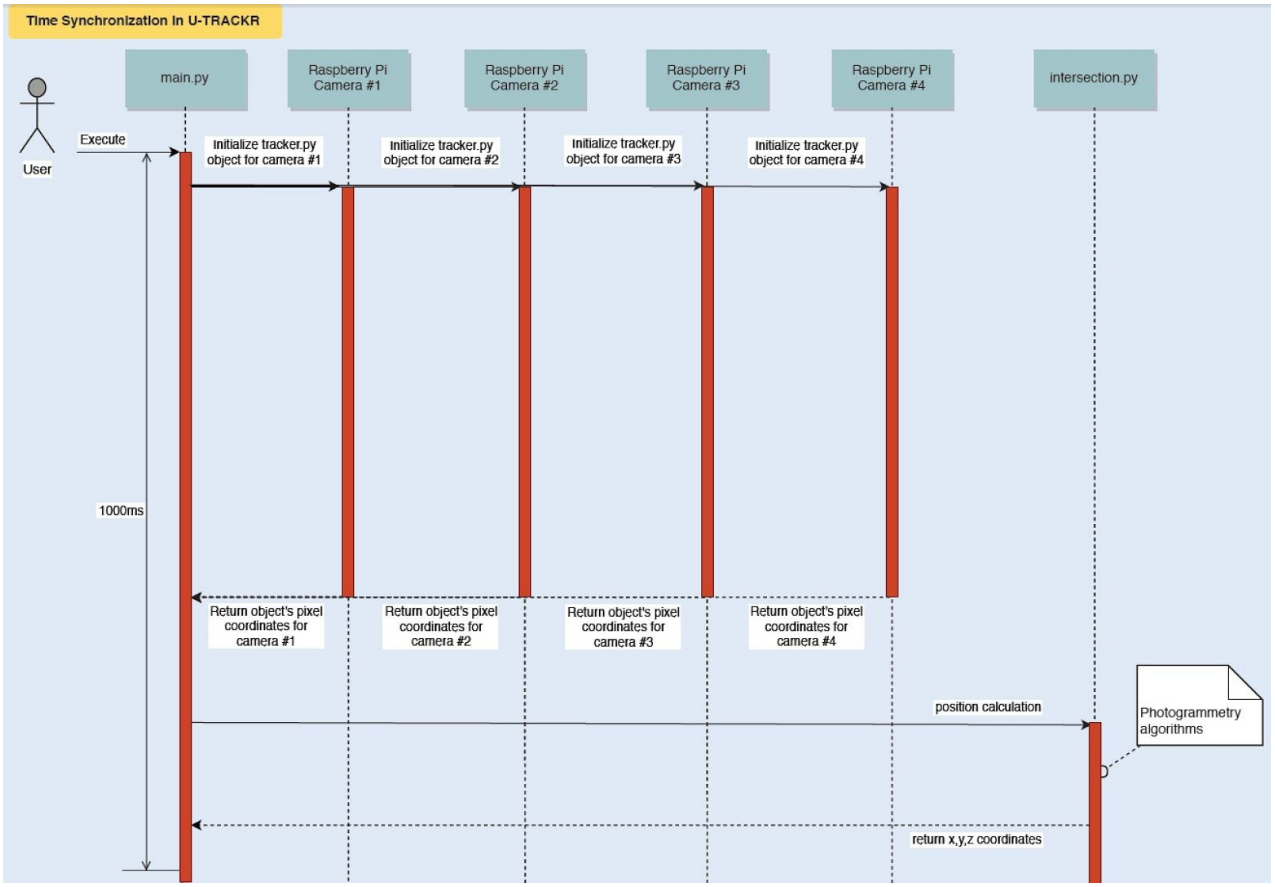


Figure 4. Sequence diagram illustrating time synchronization across multiple cameras

4.3 Camera Calibration

The camera calibration process involves finding the intrinsic parameters, distortions, skewness, focal length, and errors associated with the Pi cameras that are not provided in the specifications. This process is performed using the MATLAB Camera Calibration application where a checkerboard pattern is captured with the Raspberry Pi camera module V2. The output parameters may be different than the specified manufacturing parameters. The results are posted in the figure below:

1x1 cameraParameters

Property	Value
ImageSize	[2464,3280]
RadialDistortion	[0.2154,-0.4175]
TangentialDistorti...	[0,0]
WorldPoints	*35x2 double*
WorldUnits	'millimeters'
EstimateSkew	0
NumRadialDistorti...	2
EstimateTangentia...	0
TranslationVectors	*4x3 double*
ReprojectionErrors	*35x2x4 double*
RotationVectors	*4x3 double*
NumPatterns	4
IntrinsicMatrix	[2.6151e+03,0,0;0,2.6177e+03,0;1.5773e+03,1.2418e+03,1]
FocalLength	[2.6151e+03,2.6177e+03]
PrincipalPoint	[1.5773e+03,1.2418e+03]
Skew	0
MeanReprojection...	1.5946
ReprojectedPoints	*35x2x4 double*
RotationMatrices	*3x3x4 double*

Figure 5. Camera Calibration: Intrinsic parameters, radial distortion, tangential distortion, skewness, focal length.

4.4 Coordinate System Determination

4.4.1 Pixel Coordinate

The pixel coordinate system is calculated by utilizing the marker detection and recognition software. The pixel coordinates are given in (x_{pixel} ,y_{pixel}) coordinate with respect to the resolution of the image and origin at the top right corner of the image pixel image frame. For instance, an image with resolution of 3280 x 2464 will have a pixel maximum at those values. The pixel coordinates are taken per second and the data is put into the pixel-coordinate-to-image-coordinate converter.

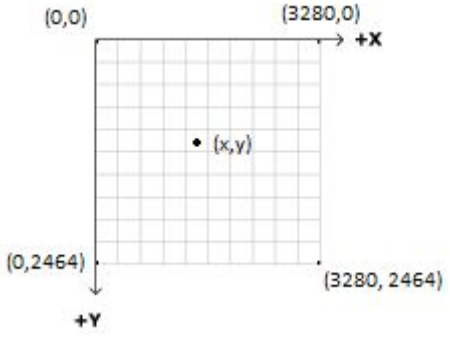


Figure 6. Pixel coordinate system

4.4.2 Image Coordinate

From pixel coordinates, the image coordinates can be found using the transformation derived from pixel size, resolution, and pixel coordinates. This process is done using the Python code, with the output matrix set aside for manipulation later. The pixel-to-image coordinate conversion is present by the following equations:

Table 3: Pixel Coordinates to Image Coordinates

Pixel Coordinates to Image Coordinates	
$SizeOfPixel = 1.12\ mm$ $Width_{ofImage} = 1280/2$ $Height_{ofImage} = 720/2$	Where, $SizeOfPixel$ is pixel size of the image predefined by the Pi cameras $Width_{ofImage}$ is predefined by the image resolution of 1280 X 720 $Height_{ofImage}$ is predefined by the image resolution of 1280 X 720
$x_{ImageCoords} = (x_{PixelCoords} - Width_{ofImage} - 0.5) * SizeOfPixel$ $y_{ImageCoords} = (Height_{ofImage} - y_{PixelCoords} + 0.5) * SizeOfPixel$	$x_{PixelCoords}$ is recorded from positional tracking algorithm presented in CDR $y_{PixelCoords}$ is recorded from positional tracking algorithm presented in CDR $x_{ImageCoords}$ is the converted X image coordinates matrix $y_{ImageCoords}$ is the converted Y image coordinates matrix
$ImageCoordinates\ (x_{ImageCoords}, y_{ImageCoords})$	$ImageCoordinates\ (x_{ImageCoords}, y_{ImageCoords})$ is the converted (x, y) image coordinates matrix $[n\ X\ 2]$
- Generic calculation of the image coordinates.	

The computed image coordinates are measured in millimeters, and the output is defined by (x_{image},y_{image}). The resultant output is used to define the parameters of the space resection calculation.

4.4.3 3D Space Coordinate

The 3D space coordinate defines the position of the object with respect to the frame. The frame is measured in meters and is defined by the (X,Y,Z) coordinate system. Figure 7 shows the relationship of positions with respect to each cameras.

4.5 Space Resection

Space resection is the mathematical computation of the camera coordinates that are used to define the frame of reference in a three-dimensional system. This process calculates the exterior orientation derived from collinearity equation, conformal transformation, linearization by means of Taylor series of expansion and least square adjustment to eliminate the blunders. The output is an iterative solution where each camera is given a 3-dimensional space coordinate and angular orientation with respect to the camera frame system.

The following equations represent the procedural relationship of the space resection process:

Table 4: Space Resection (1/4)

Space Resection (1/4)	
Exterior Orientation $(\omega, \phi, \kappa, X_L, Y_L, Z_L)$ *Image Coordinates* (x, y) *Ground Coorinates* (X, Y, Z)	Where, $(\omega, \phi, \kappa, X_L, Y_L, Z_L)$ are the exterior orientation to be calculated by space resection (x, y) are the image coordinates denoted by the lower case (X, Y, Z) are the ground coordinates denoted by the upper case
$\omega = 0, \phi = 0$	$\omega = 0, \phi = 0$ are the pre-set angular orientation to be iterated
$(\overline{AB})^2 = (X_A - X_B)^2 + (Y_A - Y_B)^2$	$(\overline{AB})^2$ is the distance calculation between one ground coordinate and the other in a planimetric coordinate
$(\overline{AB})^2 = \left(x_a\left(\frac{H - Z_A}{f}\right) - x_b\left(\frac{H - Z_B}{f}\right)\right)^2 + \left(y_a\left(\frac{H - Z_A}{f}\right) - y_b\left(\frac{H - Z_B}{f}\right)\right)^2$	H is the height to be rearranged and solved
$H = Z_L$	Z_L The height of the exterior orientation
$X'_N = x_n\left(\frac{H - Z_N}{f}\right)$ $Y'_N = x_n\left(\frac{H - Z_N}{f}\right)$	(X'_N, Y'_N) are the ground coordinates of the ground control points from the assumed vertical photo; N and n denotes index of matrix from 1 to n
$X = aX' - bY' + T_x$ $Y = bX' - aY' + T_y$	(a, b) are the coefficients of polynomial of conformal coordinate transformation using least square principle T_x, T_y are the translation of a conformal coordinate transformation using least square principle
- Space resection by iteration of initial parameters. Ground coords. [m], Image Coords. [mm]	

Table 5: Space Resection (2/4)

Space Resection (2/4)	
And the LSA solution $L = \begin{bmatrix} X_1 \\ Y_1 \\ \dots \\ X_N \\ Y_N \end{bmatrix}$	Where, L is the ground coordinate (X, Y) matrix
$A = \begin{bmatrix} X_A' & Y_A' & 1 & 0 \\ X_B' & Y_B' & 0 & 1 \\ \dots & \dots & \dots & \dots \\ X_N' & Y_N' & 0 & 1 \end{bmatrix}$	A is the designed matrix of the calculated ground control points
$\hat{X} = \begin{bmatrix} \hat{a} \\ \hat{b} \\ \widehat{T_X} \\ \widehat{T_Y} \end{bmatrix}$	$\hat{X}$ is the adjusted matrix of the least square adjustment (LSA) solution
$\hat{X} = (A^T A)^{-1} A^T L$	
$\kappa = \theta = \tan^{-1}(\frac{\hat{b}}{\hat{a}})$	κ is one of the calculated angular orientation
The rotation matrix $m = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$	m is rotation matrix used to calculate the conformal transformation of the coordinate system
$m_{11} = cos\phi cos\kappa$ $m_{12} = sin\omega sin\phi cos\kappa + cos\omega sin\kappa$ $m_{13} = -cos\omega sin\phi cos\kappa + sin\omega sin\kappa$ $m_{21} = -cos\phi sin\kappa$ $m_{22} = -sin\omega sin\phi sin\kappa + cos\omega cos\kappa$ $m_{23} = cos\omega sin\phi sin\kappa + sin\omega cos\kappa$ $m_{31} = sin\phi$ $m_{32} = -sin\omega cos\phi$ $m_{33} = cos\omega cos\phi$	m_{nn} is the individual matrix calculation of the conformal rotation
Linearization using Taylor series of Expansion Solving for $(d\omega, d\phi, d\kappa, dX_L, dY_L, dZ_L)$	$(d\omega, d\phi, d\kappa, dX_L, dY_L, dZ_L)$ are the estimated and linearized parameters using Taylor series of Expansion
$J_N = b_{n1}^N d\omega + b_{n2}^N d\phi + b_{n3}^N d\kappa - b_{n4}^N dX_L - b_{1\kappa}^N dY_L - b_{1\kappa}^N dZ_L$ $K_P = b_{n1}^P d\omega + b_{n2}^P d\phi + b_{n3}^P d\kappa - b_{n4}^P dX_L - b_{n\kappa}^P dY_L - b_{n\kappa}^P dZ_L$	J_N, K_N linearized parameter for ground coordinates (X, Y); N and P are the indices for ground coordinates from X to Y from A to D $(i.e. b^A \dots b^D)$ and n is numerical index from 1-2 row matrix $i.e. b_{11}^A \dots b_{21}^D)$
- Space resection by iteration of initial parameters	

Table 6: Space Resection (3/4)

Space Resection (3/4)

Elements for the Designed B Matrix

$$r = m_{11}(X_A - X_L) + m_{12}(Y_A - Y_L) + m_{13}(Z_A - Z_L)$$

$$s = m_{21}(X_A - X_L) + m_{22}(Y_A - Y_L) + m_{23}(Z_A - Z_L)$$

$$q = m_{31}(X_A - X_L) + m_{32}(Y_A - Y_L) + m_{33}(Z_A - Z_L)$$

$$b_{11}^{N} = \frac{f}{q^2}[r(-m_{33}\Delta Y + m_{32}\Delta Z) - q(-m_{13}\Delta Y + m_{12}\Delta Z)]$$

$$b_{12}^{N} = \frac{f}{q^2}[r(cos\phi\Delta X + sin\omega sin\phi\Delta Y - cos\omega sin\phi\Delta Z) - q(-sin\phi cos\kappa\Delta X + sin\omega cos\phi cos\kappa\Delta Y - cos\omega cos\phi cos\kappa\Delta Z)]$$

$$b_{13}^{N} = -\frac{f}{q}(m_{21}\Delta X + m_{22}\Delta Y + m_{23}\Delta Z)$$

$$b_{14}^{N} = \frac{f}{q^2}(rm_{31} - qm_{11})$$

$$b_{15}^{N} = \frac{f}{q^2}(rm_{32} - qm_{12})$$

$$b_{16}^{N} = \frac{f}{q^2}(rm_{33} - qm_{13})$$

$$b_{21}^{P} = \frac{f}{q^2}[s(-m_{33}\Delta Y + m_{32}\Delta Z) - q(-m_{13}\Delta Y + m_{12}\Delta Z)]$$

$$b_{12}^{P} = \frac{f}{q^2}[s(cos\phi\Delta X + sin\omega sin\phi\Delta Y - cos\omega sin\phi\Delta Z) - q(-sin\phi cos\kappa\Delta X + sin\omega cos\phi cos\kappa\Delta Y - cos\omega cos\phi cos\kappa\Delta Z)]$$

$$b_{23}^{P} = \frac{f}{q}(m_{11}\Delta X + m_{12}\Delta Y + m_{13}\Delta Z)$$

$$b_{24}^{P} = \frac{f}{q^2}(sm_{31} - qm_{11})$$

Where,
r, s, q are the calculated coefficients corresponding to A matrix

$b_{nn}^{N/P}$ is the element of the calculated B matrix derived from r, s, q matrix, rotation matrix, initial assignment and calculation of (ω, ϕ, κ)

- $b_{nn}^{N/P}$ are the individual elements assigned in the B matrix.

Table 7: Space Resection (4/4)

Space Resection (4/4)	
$b_{25}^{P} = \frac{f}{q^2}(sm_{32} - qm_{12})$ $b_{26}^{P} = \frac{f}{q^2}(rm_{33} - qm_{13})$	Where,
Least Square Adjustment of B matrix $B = \begin{bmatrix} b_{11}^{A} & b_{12}^{A} & b_{13}^{A} & -b_{14}^{A} & -b_{15}^{A} & -b_{16}^{A} \\ b_{21}^{A} & b_{22}^{A} & b_{23}^{A} & -b_{24}^{A} & -b_{25}^{A} & -b_{26}^{A} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ b_{23}^{D} & b_{23}^{D} & b_{23}^{D} & -b_{24}^{D} & -b_{25}^{D} & -b_{26}^{D} \end{bmatrix}$	B matrix is the design matrix to solve the calculate exterior orientation matrix; derived from the combined elements of b matrix
$L = \begin{bmatrix} X_1 \\ Y_1 \\ \dots \\ X_N \\ Y_N \end{bmatrix}$	L is the ground coordinate (X, Y) matrix
$\Delta = \begin{bmatrix} d\omega \\ d\phi \\ d\kappa \\ dX_L \\ dY_L \\ dZ_L \end{bmatrix}$ $\Delta = (B^T B)^{-1}(B^T L)$	Δ solution of the LSA of B matrix
Calculation of Exterior Orientation (EO) $(\omega, \phi, \kappa, X_L, Y_L, Z_L)$ $\omega = d\omega\left(\frac{180^o}{\pi}\right) \pm 360^o$ $\phi = d\phi\left(\frac{180^o}{\pi}\right) \pm 360^o$ $\kappa = \theta + d\kappa$ $X_L = T_X + dX_L$ $Y_L = T_Y + dY_L$ $Z_L = T_Z + dZ_L$	$(\omega, \phi, \kappa, X_L, Y_L, Z_L)$ calculated E.O
- Calculated E.O. in $[m]$ and $[^o]$	

4.6 Space Intersection

Space intersection uses the principle of stereopair image to find the intersection of two rays that connect through a pair of planes. The process of space intersection finds the positional coordinates of an object given the exterior orientations of the camera system.

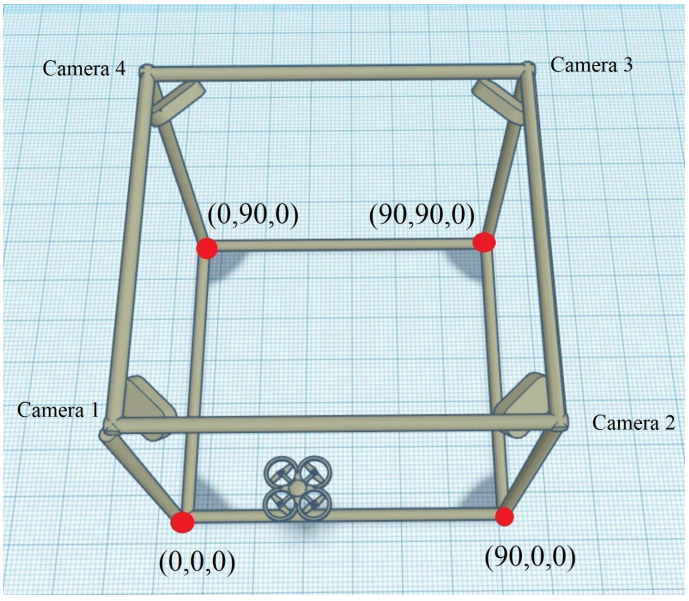


Figure 7. Camera frame system: origin of the system is defined at the corner below camera one.

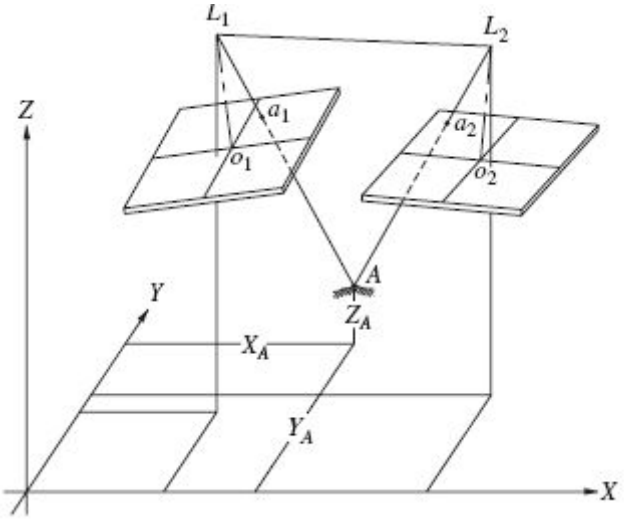


Figure 8. Stereopair property

The final 3D-space coordinate solution is an iterative process similar to space resection, where collinearity equations are used with the least square solution. The accuracy of the solution is dependent on the number of cameras where an increase in number of cameras increases the degrees of freedom, thus enhancing the results. The law of diminishing returns applies when there is an excess amount of cameras but the accuracy does not increase significantly.

The following equations represent the procedural relationship of the space intersection process:

Table 8: Space Intersection

Space Intersection	
Elements for the Designed B Matrix $b_{14}^{N} = \frac{f}{q^2}(rm_{31} - qm_{11})$ $b_{15}^{N} = \frac{f}{q^2}(rm_{32} - qm_{12})$ $b_{16}^{N} = \frac{f}{q^2}(rm_{33} - qm_{13})$ $b_{24}^{P} = \frac{f}{q^2}(sm_{31} - qm_{11})$ $b_{25}^{P} = \frac{f}{q^2}(sm_{32} - qm_{12})$ $b_{26}^{P} = \frac{f}{q^2}(rm_{33} - qm_{13})$	Where, $b_{nn}^{N/P}$ is the element of the calculated B matrix derived from r, s, q matrix, rotation matrix, initial assignment and calculation of (ω, ϕ, κ)
Least Square Adjustment of B matrix $B = \begin{bmatrix} -b_{14}^{A} & -b_{15}^{A} & -b_{16}^{A} \\ -b_{24}^{A} & -b_{25}^{A} & -b_{26}^{A} \\ \cdots & \cdots & \cdots \\ -b_{24}^{N} & -b_{25}^{N} & -b_{26}^{N} \end{bmatrix}$	B matrix is the design matrix to solve the calculate exterior orientation matrix; derived from the combined elements of b matrix
$L = \begin{bmatrix} X_1 \\ Y_1 \\ \cdots \\ X_N \\ Y_N \end{bmatrix}$	L is the ground coordinates (X, Y) matrix with respect to image coordinates
$\Delta = \begin{bmatrix} dX_L \\ dY_L \\ dZ_L \end{bmatrix}$	
$\Delta = (B^T B)^{-1}(B^T L)$	Δ solution of the LSA of B matrix in (X_L, Y_L, Z_L) solution
Iterating Parameters $X_L = dX_L + X_L$ $Y_L = dY_L + Y_L$ $Z_L = dZ_L + Z_L$	Iterative solution of (X_L, Y_L, Z_L)
- $b_{nn}^{N/P}$ are the individual elements assigned in the B matrix.	

5.0 System Integration

The following tables provide a summary of the models and versions of hardware and software needed for the project.

Table 9: Summary of software frameworks required

Software	Version
Python	2.7.13
OpenCV	3.4.0
MATLAB	9.3
U-TRACKR Codebase	beta

Table 10: Summary of hardware required

Hardware
Raspberry Pi Zero Wireless (W) board with onboard WiFi and Bluetooth
Official Raspberry Pi Foundation Zero case with 3 interchangeable lids
8 GB MicroSD card (Class 10) pre-loaded with NOOBS
CanaKit 1A power supply
Raspberry Pi Camera Module V2
Anker PowerLine micro USB cable (10ft)
Anker 40W/8A 5-port USB charger PowerPort 5
Mini HDMI adapter and USB OTG cable
Frame or fixed camera mounts

5.1 Software Architecture

The main controller acts as the client, that requests services to determine the position of the objects within the frame. One of our main requirements, which we identified as a requirement in the Critical Design Review, was the following: “Synchronizing all four cameras to determine the accurate location of the objects”. The system uses the Python threading module and Paramiko framework to establish a secure shell protocol with all four Raspberry Pi modules and concurrently output each camera’s timestamp. This will synchronize the frames of one camera to the other, allowing for consistent data. The main controller will then process the camera image frames using the OpenCV software to identify objects and output the pixel coordinates relative to the frame. Space resection computation will be performed on the coordinates to define the frame of reference in a 3D system.

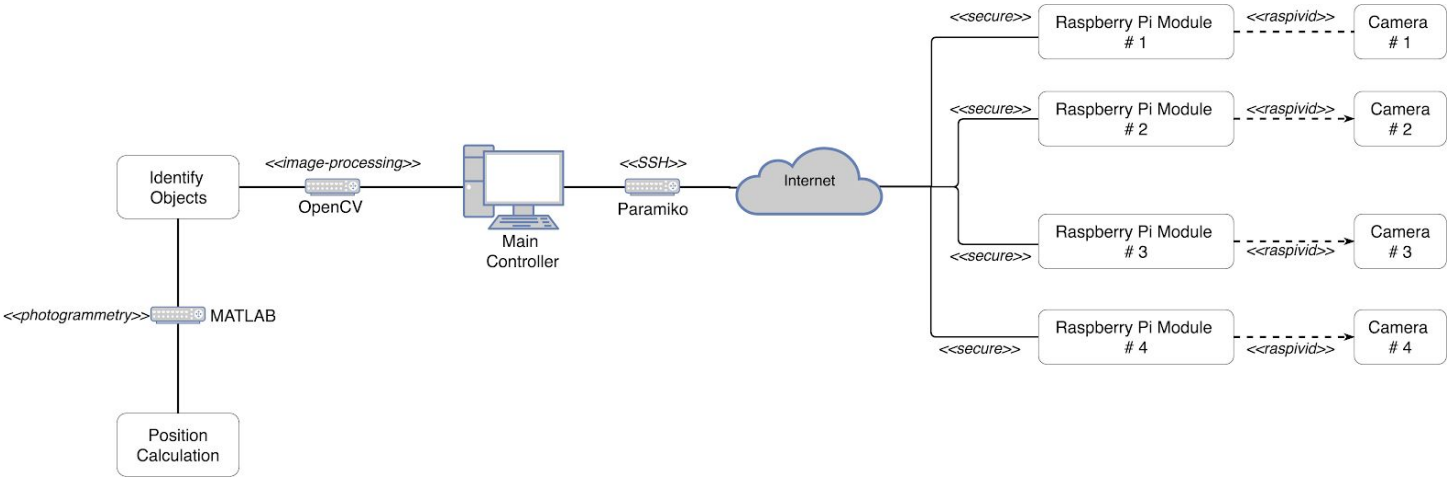


Figure 9. Software architecture diagram of the U-TRACKR system

5.2 Physical Design

The initial system design was inadequate in keeping the cameras at fixed positions, which is critical in the space resection and intersection calculations. Moreover, the initial frame proved to be problematic since it deterred the system from obtaining correct coordinate results. Furthermore, the camera holders were not easily adjustable.

Figure 10. Initial system frame and camera holders

The physical design of the system frame and camera holders in the U-TRACKR system was upgraded to improve system stability and consistency. The new designs made it easier to change the camera angles and the designs improved the overall system appearance.

5.2.1 System Frame

In previous designs, the camera angles changed at the slightest disturbances of the frame. Therefore, the team obtained steel beams to increase system stability. A metal foam board was used for the base, and a grid was created with known coordinates for camera calibration.

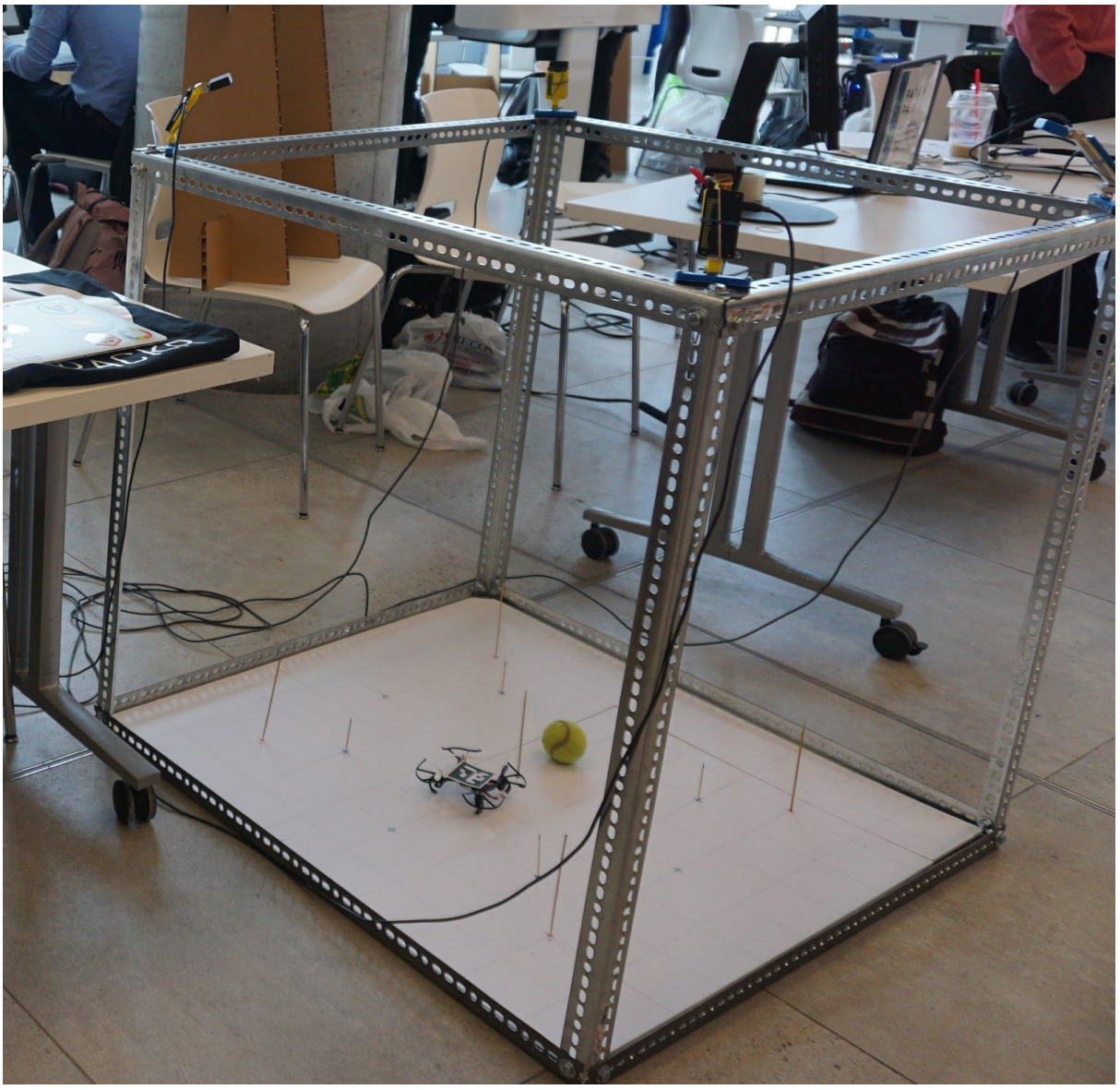
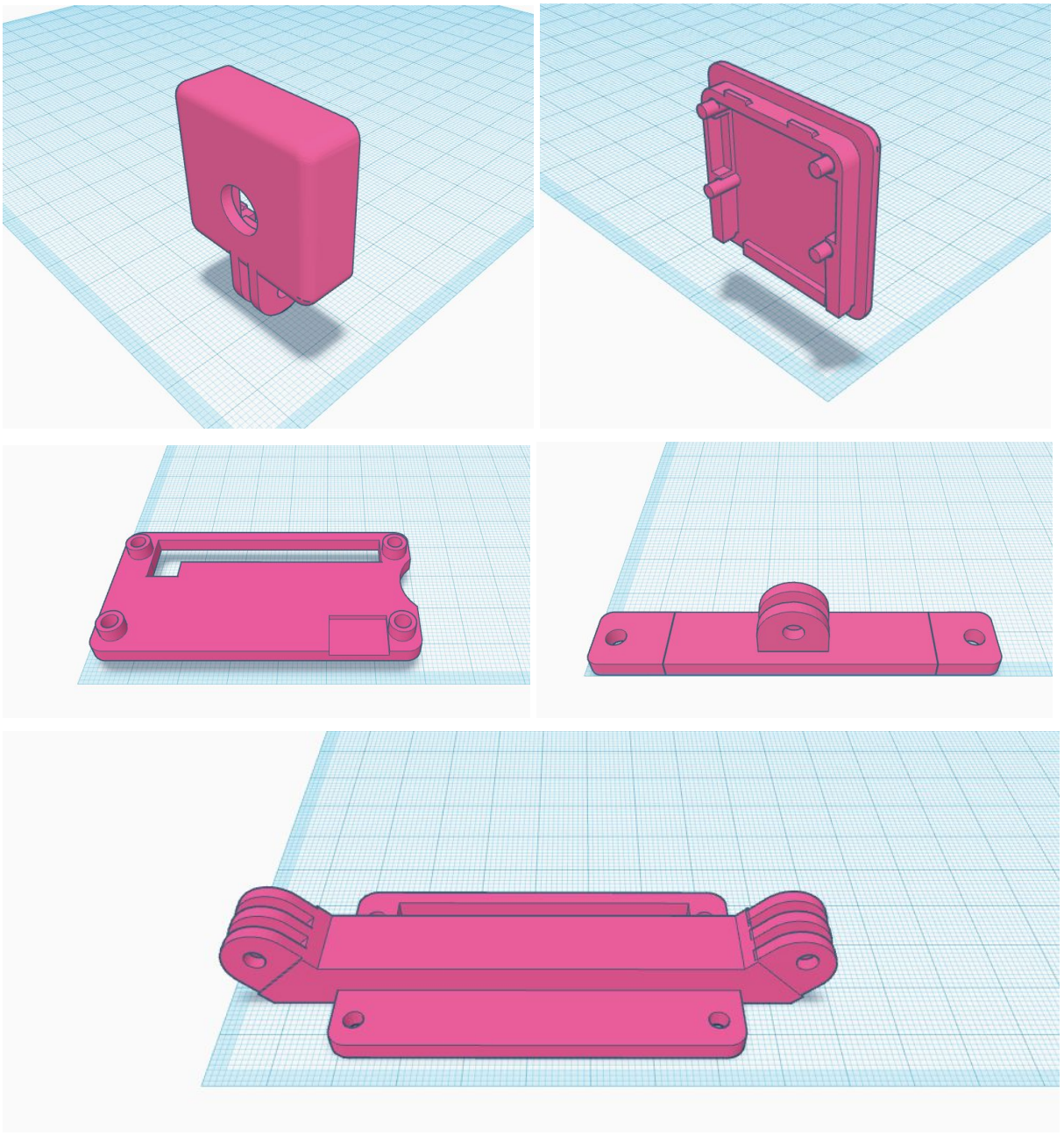


Figure 11. Final physical design of U-TRACKR system

5.2.2 Digital Design for Camera Modules

The camera module holders were 3D-printed at the Lassonde lab and some of the designs were obtained from Thingiverse. The mounts were created in CAD software to securely attach to the the U-TRACKR system frame.



Figures 12a, 12b, 12c, 12d, 12f. Digital designs for U-TRACKR camera module holders

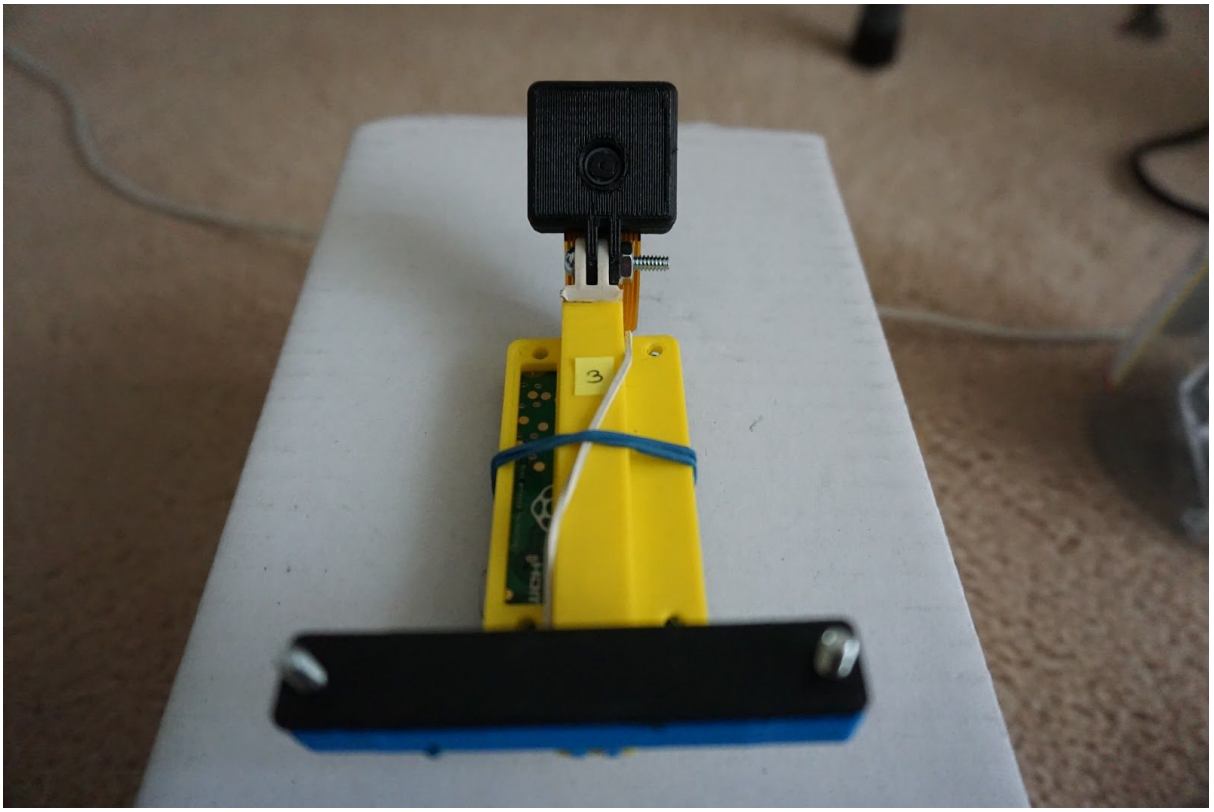


Figure 13a, 13b. Final physical design of U-TRACKR camera module holders

5.3 Software Design

5.3.1 Software Classes

There are four main Python classes used to run the U-TRACKR system.

Table 11: Python classes

Class	Description
timesync.py	Connects the main controller to the Raspberry Pi using the Paramiko library by establishing an SSH client connection. Has various methods to start a continuous video stream, and obtain the current timestamp of the Raspberry Pi.
tracker.py	Performs computer vision functions on each image frame provided by the video stream. It stores the pixel coordinates of the tracked objects.
intersection.py	Performs a position calculation given the exterior camera orientation parameters, and the pixel coordinates of the tracked object as input.
main.py	Starts each Raspberry Pi video stream concurrently using threads, and obtains a position calculation at every sampling instance.

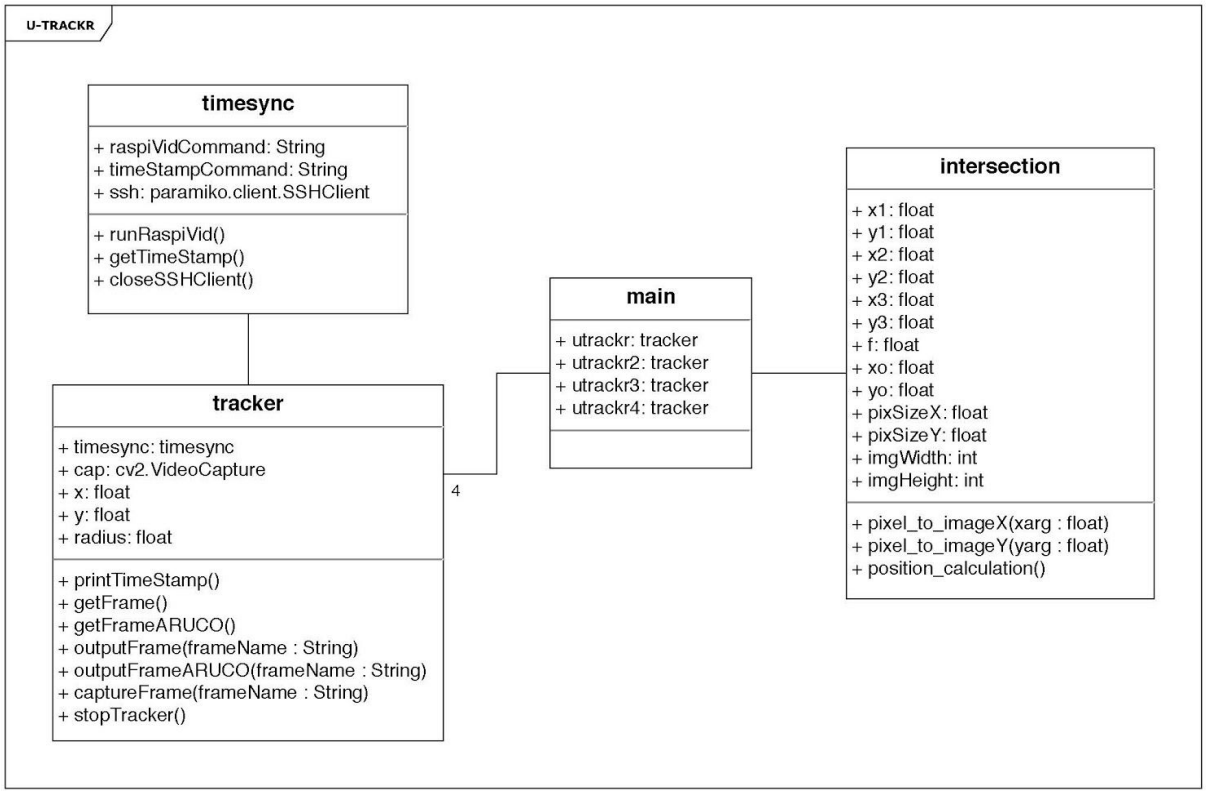


Figure 14. UML Class Diagram of U-TRACKR system

5.3.2 Software Integration

The figure below depicts the workflow of a user executing the U-TRACKR system. The entry point of this system is the execution of the main.py program which enables and starts all background services such as tracker.py, timesync.py, cv2, and checker.py in order to retrieve the X, Y and Z coordinates of the object placed within the frame.

When main.py begins, its initial step is to retrieve synchronized camera frames from all four Raspberry Pi camera modules. Using these frames, the program performs image processing techniques using the OpenCV library to identify objects and determine their X, and Y coordinates relative to the image. Finally, the program performs space resection and intersection using the extracted X and Y coordinates to produce the X, Y, Z coordinates of the object.

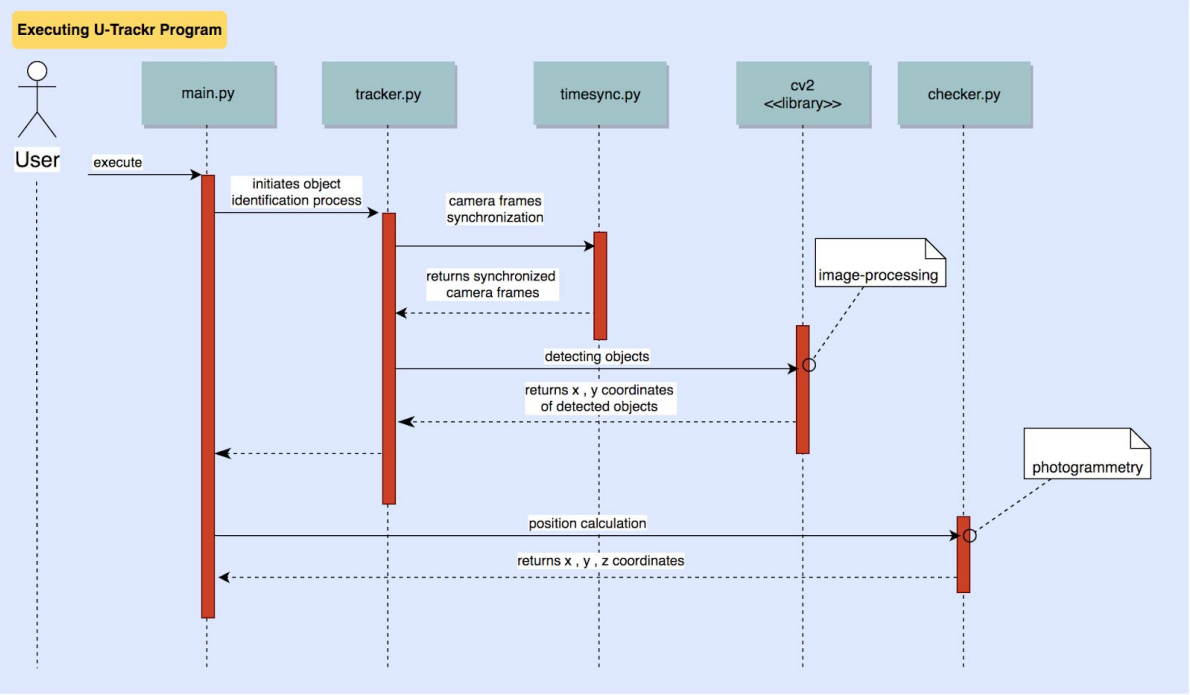


Figure 15. Sequence diagram of executing the U-TRACKR system

6.0 System Performance

6.1 System Results

In this section, the results from the space resection and space intersection are analyzed.

6.1.1 Space Resection Results

The exterior orientation parameters of camera one uses the 13 ground control points (X,Y,Z) coordinate marked in the image below. Each GCP is associated with a pixel coordinate value measured at a resolution of 3280X2464. The pixel coordinates are then converted into image coordinates. When the image resection function or *single_photo_resection.m* in *MATLAB* is run, the camera position with respect to the frame is found. (The input format can be found in Appendix A).

Figure 16. Space Resection performed on MATLAB for camera one image

The iterative solution takes an initial exterior orientation or initial looping parameters of 3D space coordinate and angular orientation. The iteration stops until the error produced by the least square adjustment is negligible (less than 0.00000001 m or 0.00001 mm). These numbers are changed accordingly for each camera as they occupy coordinate space and angular orientation. For instance, the following example is produced by camera one after 20 iterations.

Table 12: Space Resection Results: Iterative solution

Camera One Image Resection	**Number of iterations**	**Space coordinates** (X,Y,Z) [m]	**Angular Orientation** (ω,ϕ,κ) [rads]
Initial Looping Parameter	0	(0.05, 0.05, 0.98)	(0.785, -0.785, 0.0)
Negligible LSA (LSA > 0.0000001)	13	(0.0745, 0.0783, 1.0686)	(0.3516, -0.3403, -2.3199)
Resection Set Counter	20	(0.0745, 0.0783, 1.0686)	(0.3516, -0.3403, -2.3199)

After the image resection is run, two outcomes occur: the LSA becomes negligible when it meets the tolerance, or the program runs 20 iterations. The negligible solution shows us that the results converge after 13 iterations with a tolerance of 0.00001 mm (or 0.00000001 m) error. Therefore, after 13 iterations, the solution converges to tolerance and the iterative process becomes unnecessary, by law of diminishing returns.

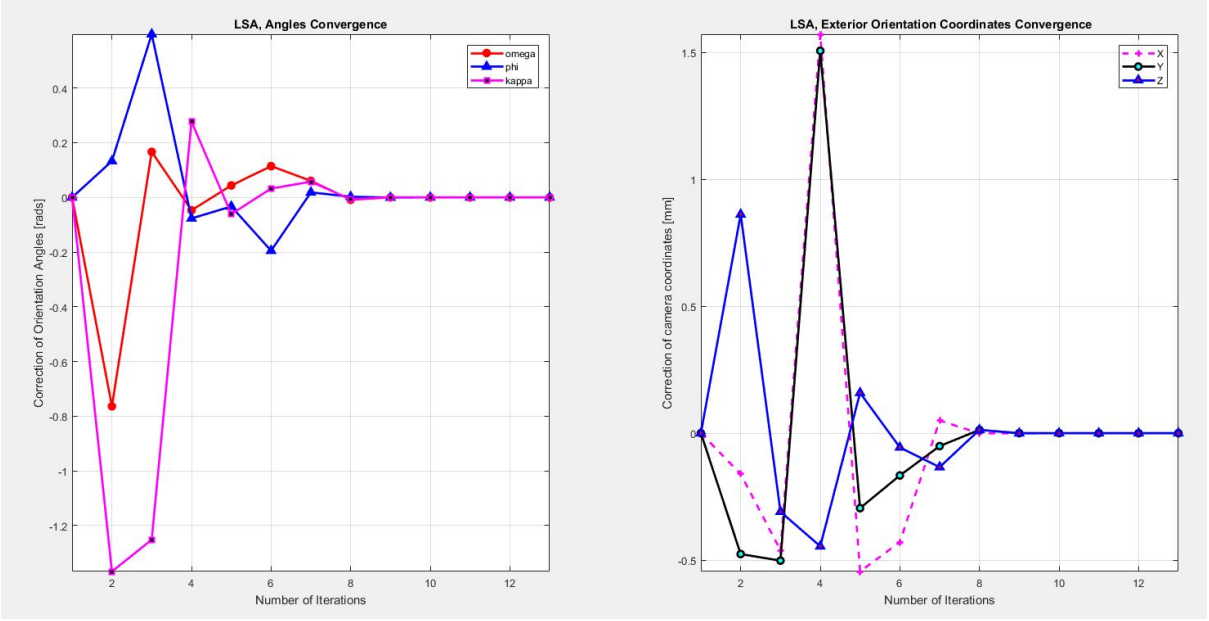


Figure 17. Convergence Analysis: angles convergence (left) after 13 iterations, and exterior orientation coordinates convergences (right) after 13 iterations.

The output exterior orientation values are subject to the error in the frame coordinate and the residual from iterative solutions. The iterative solutions are negligible since the system cannot produce a tolerance of 0.00001 mm accuracy. Therefore most of the error occurs between where GCPs are defined and the true values of GCPs with respect to the frame.

6.1.2 Space Intersection Results

Space intersection uses the exterior orientation of all camera parameters calculated from space resection. These six parameters for each camera are the inputs for intersection calculation of a stereopair.

The inputs are compiled in the *single_Intersection.m* method and the outputs are produced in space coordinates (X,Y,Z) with respect to the frame (Refer to Appendix B for sample input). The iterative solution stops when the LSA becomes negligible or when it encounters a set loop number. For instance, the diagram below shows a drone position near the center of the frame.

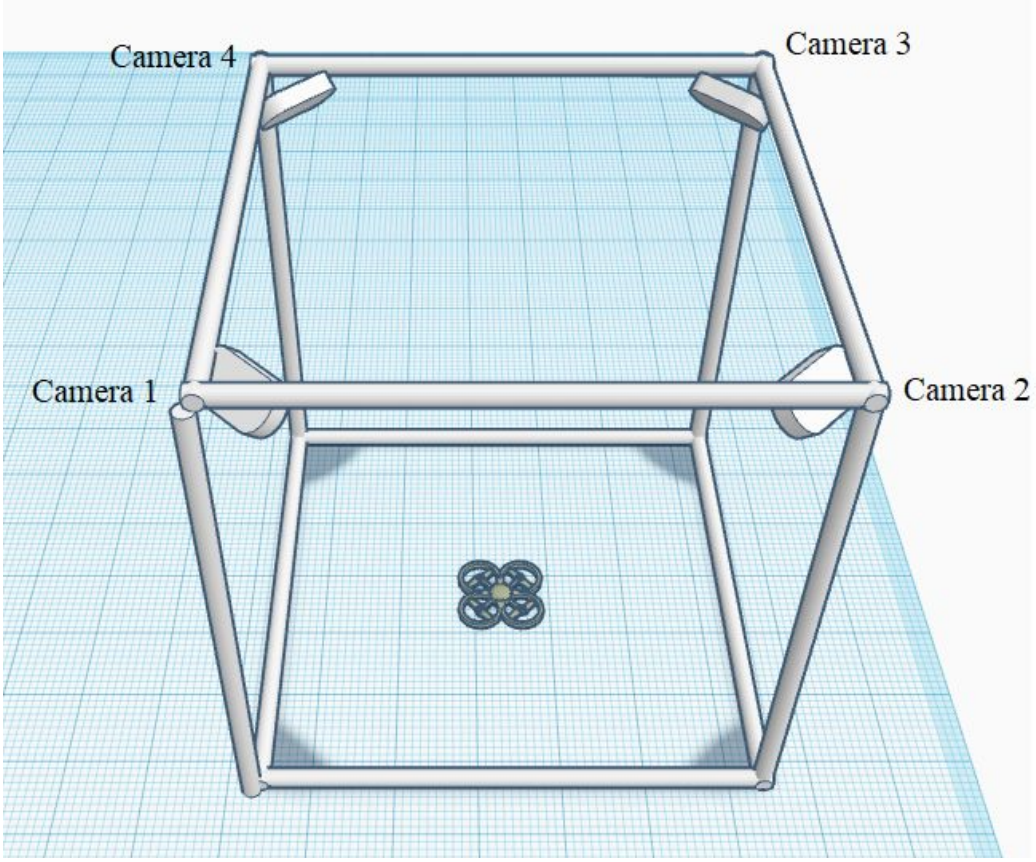


Figure 18. Space Intersection: object close to the frame center

The expected position of the drone is an approximation made by measuring the (X,Y,Z) of the drone with respect to the frame system. This is the value of expected output. The measurement accuracy of the ruler has a value of ± 0.5 cm or ± 5.0 mm plus the error in the system frame.

Table 13: Space Intersection Results: Iterative solution.

Cameras (1-3) Image Intersection	**Number of Iterations**	**Space Coordinates** (X,Y,Z) [m]
Expected Output	0	(0.40, 0.40, 0.0625)
Negligible LSA (LSA > 0.0000001)	7	(0.4190,0.4115, 0.0915)
Intersection Set Counter	20	(0.4190,0.4115, 0.0915)

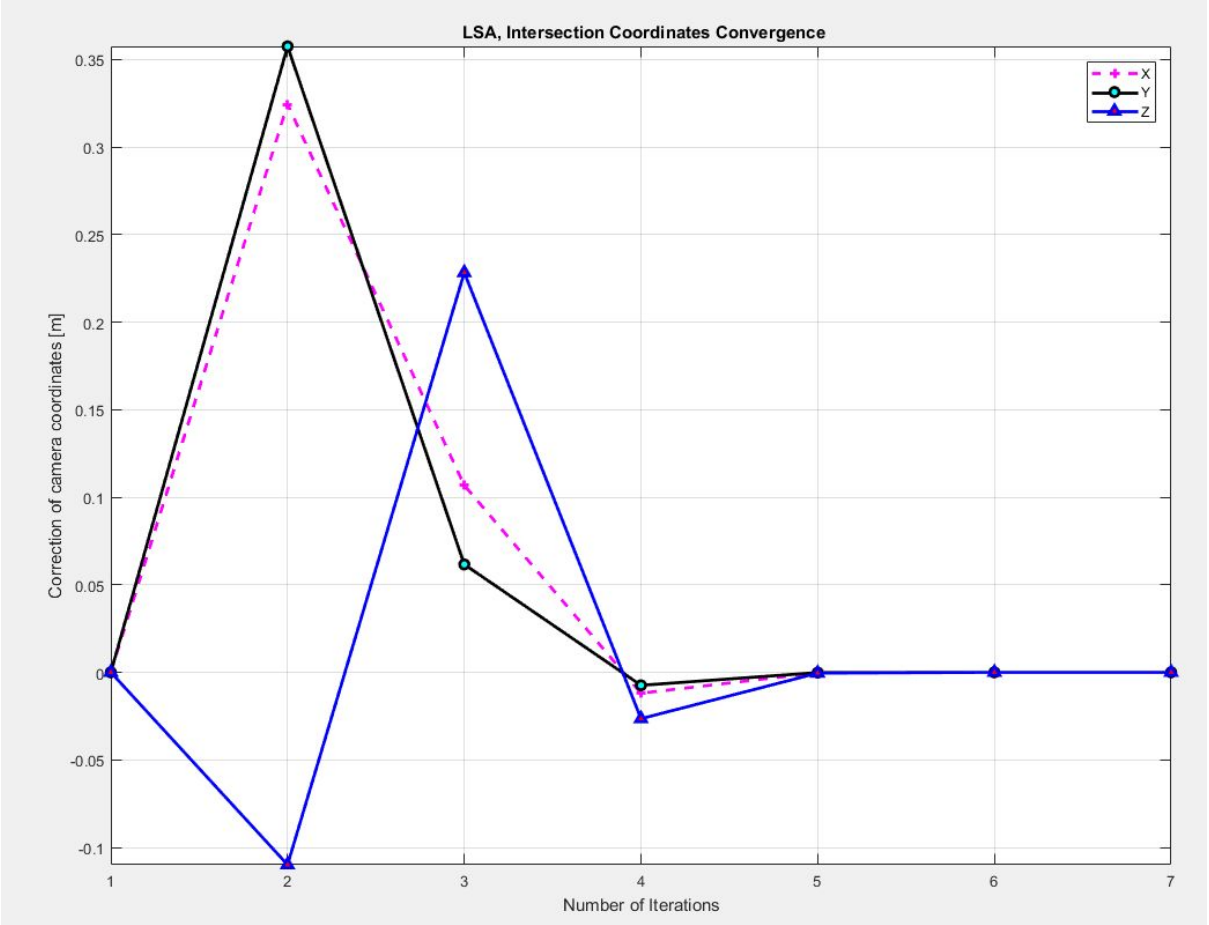


Figure 19. Convergence Analysis: LSA intersection coordinates convergences after 7 iterations.

After the image intersection is run, two outcomes occur: the LSA becomes negligible when it meets the tolerance, or the programs runs 20 iterations. The negligible solution shows us that the results converge after 7 iterations with a tolerance of 0.00001 mm (or 0.00000001 m) error. Therefore, after 7 iterations, the solution converges to tolerance and the iterative process become unnecessary, by law of diminishing returns.

The output intersection values are subject to the error in the frame coordinate and the residual from iterative solutions. Similar results were produced with the translated Python code *intersection.py.* The iterative solutions are negligible since the system cannot produce a tolerance of 0.00001 mm accuracy. Therefore most of the error occurs between where GCPs are defined and the true values of GCPs with respect to the frame.

6.2 System Testing

System testing was performed to verify whether the U-TRACKR system is fit for its intended purposes. The system was evaluated by verification tests which covered eight areas. Overall, The U-TRACKR system passed 13 out of 15 test cases, and the failures are explained below.

Table 14: System Test Results

Test ID	Results	Description	Future Improvements
FUNC-01	PASS	Verify that each Raspberry Pi can connect to the same WiFi network as the main controller (laptop)	None
FUNC-02	PASS	Verify the ability to establish an SSH connection from the main controller to each Raspberry Pi.	None
FUNC-03	PASS	Verify the ability to stream the camera feed from the Raspberry Pi's to the main controller from each Raspberry Pi's at the same time.	None
FUNC-04	PASS	Verify that OpenCV software runs with no issues on main controller and identifies objects within the frame.	None
FUNC-05	PASS	Verify that the U-TRACKR program executes the Python script that determines the X, Y, Z coordinates of the object within in the frame.	None
PER-01	PASS	Check if each Raspberry Pi CPU usage meets the specified requirements.	None
PER-04	PASS	Check if each Raspberry Pi stream bitrate meets the specifications.	None
MTBF-01	FAIL	Determine the mean time before failure of the U-TRACKR system while running for at least 8 hours.	Perform error handling to prevent exceptions regarding matrix calculations in intersection.py.
ENV-01	FAIL	Determine the U-TRACKR program is functional within a specific lux threshold	Use machine learning models by training the model to accurately detect the object in low light without the need of markers.
REL-01	PARTIAL PASS	Check if the U-TRACKR program will not crash if one or more of the Raspberry Pi camera breaks down.	Perform error handling in the case where one camera breaks down. Stop the system at the point where less than 2 cameras are operating, or the 2 cameras remaining are perpendicular to each other (non-stereopair).
REL-02	PASS	Verify the accuracy of the position calculated by the U-TRACKR program.	None
COM-01	PASS	Determine that the U-TRACKR program can identify objects using the full range of values (ex. HSV colours)	None
POR-01	PASS	Determine the level of ease to apply the U-TRACKR program to other hardware applications.	None
BVT-01	PASS	Determine the U-TRACKR program is functional when tracking an object at the farthest distance specified by the requirements.	None
BVT-02	PASS	Determine the U-TRACKR program is functional when tracking an object at the shortest distance specified by the requirements.	None

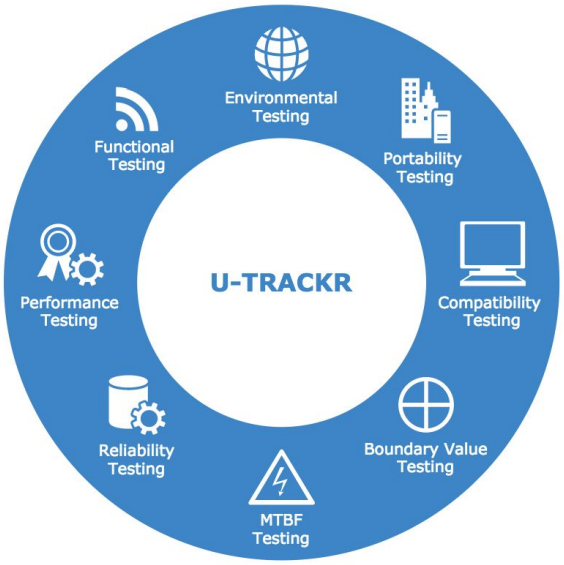


Figure 20. U-TRACKR System Testing Diagram

7.0 Deliverables And Setup

7.1 Hardware Deliverables

The hardware deliverables for U-TRACKR consist of the following components:

Table 15: List of hardware components and a description of their uses

Item	Description
Raspberry Pi Zero W Starter Kit with Case (x4)	Controls and processes information from the camera modules.
Raspberry Pi Camera Module V2 (x4)	Takes high-definition videos in low-latency time for computer vision based tracking.
Primary Processing System (PC)	Tracks, and locates the moving object(s) programmatically using image sequences. An ideal processing system would have a hardware configuration similar to the following: • *Operating System*: Windows 10 Home 64-bit (10.0, Build 15063) • *Processor*: Intel(R) Core(™) i7-7700HZ CPU @ 2.80GHz (8 CPUs) • *Memory*: 16384MB RAM • *Display Adapter*: Intel(R) HD Graphics 630
Frame	Holds the camera modules in opposite and equidistant positions at each corner of the frame. An ideal frame would have an equal side length of around 90cm and be sturdy enough to hold each Raspberry Pi Zeros without disrupting the position and angle of the camera.
Power Supply for the Raspberry Pi Zeros	Powers the Raspberry Pi Zeros during operation of the U-TRACKR system.An ideal power supply system would be able to reach each corner of the frame without disrupting the position and angle of each camera.

7.2 Software Deliverables

The software deliverables for the Primary Processing System for U-TRACKR consists of the following components:

Table 16: List of software necessary for the project and a description for each program

Item	Description
Python 2.7.13	Python is chosen as the primary programming language due to its various libraries in image processing and computer vision. It is also specifically useful in interacting with the Raspberry Pi programmatically.
OpenCV 3.4.0	OpenCV is chosen as the primary software framework used to track object due to it's free availability in commercial and educational use, as well as its support with Windows and Python interfaces.
MATLAB 9.3	MATLAB is used to perform camera calibration to obtain the interior camera orientation parameters, and to run the space resection algorithm used for position calculation.
U-TRACKR Codebase	Used to run the U-TRACKR system. This could be found in the following GitHub link: https://github.com/azkevin/U-TRACKR

7.3 Setup

The following steps detail how to setup the environment for both the Raspberry Pi microcontrollers and the primary processing system.

It is assumed that the hardware deliverables are obtained, and the software deliverables are installed on the primary processing system. It is assumed that all Raspberry Pi Zero microcontrollers are powered on, and are connected to the internet using Wifi or ethernet.

1. Configure each Raspberry Pi Zero

1.1. Setup the Camera Module on each Raspberry Pi Zero

1.1.1. Upgrade the hardware firmware of the Raspberry Pi Zero by running the following commands on the terminal window:

1.1.1.1. "sudo apt-get update"

1.1.1.2. "sudo apt-get upgrade"

1.1.2. Enable the Camera Module on the Raspberry Pi Zero by following the steps:

1.1.2.1. Run "sudo raspi-config" on the terminal window > 5 Enable Camera > Enable > Reboot

1.1.3. Once rebooted, confirm that the camera is detected by running the following command and ensuring both values are 1.

1.1.3.1. "vcgencmd get_camera"

1.2. Enable SSH on each Raspberry Pi Zero

1.2.1. Enable SSH on the Raspberry Pi Zero by following the steps:

1.2.1.1. Run “sudo raspi-config” on the terminal window > Select Interfacing Options > SSH > Yes > Ok > Finish

2. Running U-TRACKR

2.1. Obtain and replace the IP Address of each Raspberry Pi Zero

2.1.1. Run the following command on the terminal window: “ifconfig”

2.1.2. Obtain and save the IP Address by looking at “wlan0 > inet addr”

2.1.3. Replace the IP Addresses found in release/main.py with the corresponding ones found during this step.

2.2. Run the Space Resection Algorithm to obtain initial parameters

2.2.1. Capture images from each Raspberry Pi Zero by running “raspicam -o image.jpg”

2.2.2. Obtain the pixel coordinates of each point on each image. Replace the values in cam_Pixel_Coords in release/single_photo_resection.m

2.2.3. Estimate the camera position ground coordinates and replace the values in x0, y0, and z0 in release/single_photo_resection.m

2.2.4. Run release/single_photo_resection.m and replace the values of cam_Ground_Control_Coords, omega, phi, kappa in release/intersection.py for each image.

2.3. Run U-TRACKR by running “python main.py” under the release folder on the terminal window.

8.0 Project Management and Finances

8.1 As-Built Project Schedule

A breakdown of scheduled tasks and resources can be found below. This can also be seen using Microsoft Project with various views by following this link.

	Task Name	Duration	Start	Finish	Predecessors
1	**U-TRACKR**	**194 days**	**Mon 10/16/17**	**Fri 4/27/18**	
2	**Preliminary Design Review (PDR)**	**8 days**	**Mon 10/16/17**	**Mon 10/23/17**	
3	PDR Draft	2 days	Mon 10/16/17	Tue 10/17/17	
4	Meeting with industry advisor	1 day	Wed 10/18/17	Wed 10/18/17	3
5	PDR Final Report	5 days	Thu 10/19/17	Mon 10/23/17	4
6	**Critical Design Review (CDR)**	**42 days**	**Tue 10/24/17**	**Mon 12/4/17**	**2**
7	Prototyping (Microcontroller and frame setup)	15 days	Tue 10/24/17	Wed 11/8/17	
8	Prototyping (Development of a preliminary object tracking program)	17 days	Wed 11/8/17	Sun 11/26/17	7
9	CDR Draft	1 day	Sun 11/26/17	Tue 11/28/17	8
10	Prototype review with technical supervisor	1 day	Wed 11/29/17	Wed 11/29/17	9
11	CDR Final Report	5 days	Thu 11/30/17	Mon 12/4/17	10
12	**Test Readiness Review (TRR)**	**64 days**	**Mon 12/4/17**	**Mon 2/5/18**	**6**
13	Microcontroller, camera, and frame setup	14 days	Mon 12/4/17	Sun 12/17/17	
14	Semester break	11 days	Sun 12/17/17	Thu 12/28/17	
15	Development of the primary object tracking algorithm	13 days	Thu 12/28/17	Thu 1/11/18	
16	Design review with technical supervisor	1 day	Fri 1/12/18	Fri 1/12/18	13,15
17	Design test plan for full system testing	8 days	Fri 1/12/18	Tue 1/23/18	16
18	TRR Draft	6 days	Tue 1/23/18	Fri 2/2/18	17
19	TRR Final Report	3 days	Sat 2/3/18	Mon 2/5/18	18
20	**Test Review (TR)**	**42 days**	**Mon 2/5/18**	**Sun 3/18/18**	**12**
21	Creation of all Python automated tests	10 days	Mon 2/5/18	Wed 2/14/18	
22	Running all automated tests	6 days	Thu 2/15/18	Tue 2/20/18	21
23	Running all manual tests	16 days	Wed 2/21/18	Thu 3/8/18	
24	TR Draft	5 days	Fri 3/9/18	Tue 3/13/18	23,22
25	TR Final Report	5 days	Wed 3/14/18	Sun 3/18/18	24
26	**Final Project Documentation (FPD)**	**29 days**	**Sun 3/18/18**	**Sun 4/15/18**	**20**
27	Final changes to system design based on testing	13 days	Sun 3/18/18	Fri 3/30/18	
28	FPD Draft	7 days	Fri 3/30/18	Fri 4/6/18	27
29	FPD Report	9 days	Thu 4/5/18	Sun 4/15/18	28
30	**Product Release Presentation/Exhibits**	**13 days**	**Sun 4/15/18**	**Fri 4/27/18**	**26**
31	Demo materials preparation (poster, videos, shirts, etc)	6 days	Sun 4/15/18	Fri 4/20/18	
32	Demo preparation practice	7 days	Fri 4/20/18	Fri 4/27/18	
33	Demonstration	0 days	Fri 4/27/18	Fri 4/27/18	32,31

Figure 21. Gantt Chart View of the Project Schedule

8.2 Work Breakdown Structure

A product-based work breakdown structure can be found below.

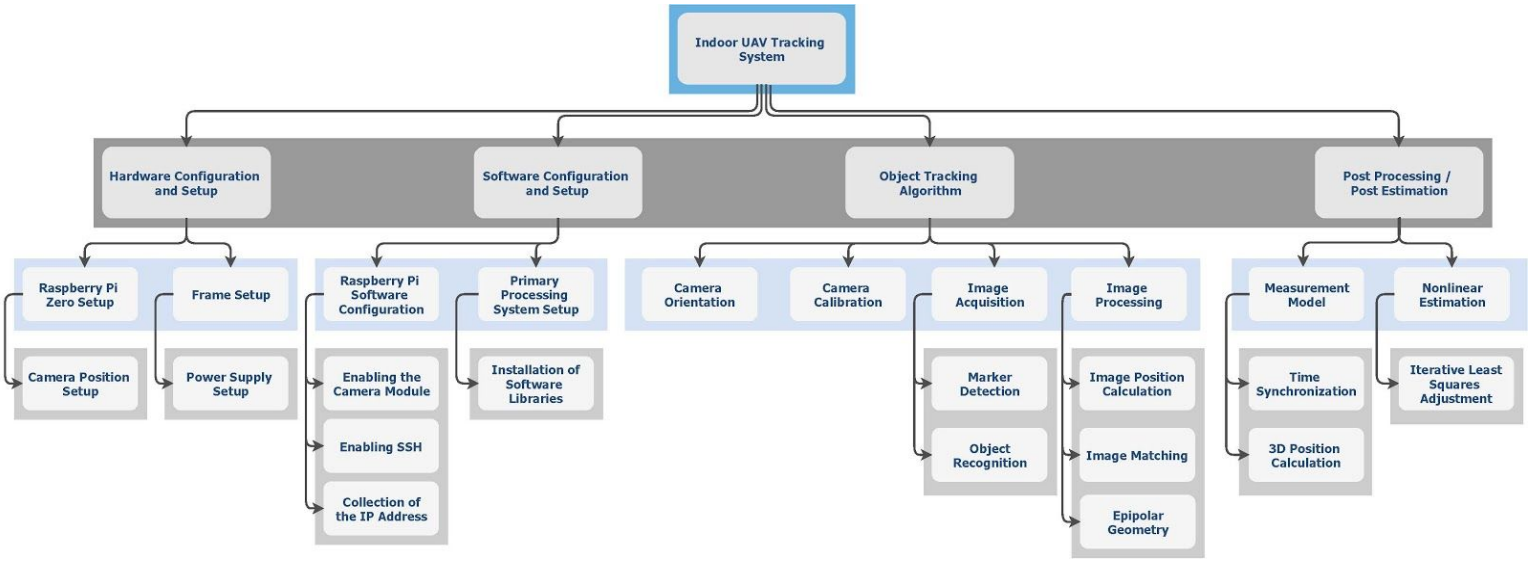


Figure 22. Work Breakdown Structure of U-TRACKR

8.3 Cost Analysis

8.3.1 Prototype

Table 17: Prototype cost

Item	Cost	Description
Raspberry Pi Zero W Starter Kit with Case (x2)	($49.99 * 2) = **$99.98**	Controls and processes information from the camera modules.
Raspberry Pi Camera Module V2 (x2)	($30.99 * 2) = **$61.98**	Takes high-definition videos in low latency for vision based tracking.
PVC Pipes	**$26.40**	Frame components to hold the camera modules in opposite and equidistant positions.
T-Connectors (x8)	**$15.04**	
45° Connectors (x16)	**$25.28**	
Dust Masks	**$2.50**	
Spray Paint	**$11.99**	
PROTOTYPE COST: $243.17		

8.3.2 Final design

Table 18: Final cost

Item	Cost	Description
Raspberry Pi Zero W Starter Kit with Case (x4)	($49.99 * 4) = **$199.96**	Controls and processes information from the camera modules.
Raspberry Pi Camera Module V2 (x4)	($30.99 * 4) = **$123.96**	Takes high-definition videos in low latency for vision based tracking.
Hillman Plated Steel Slotted Angle (x12)	**$160.56**	A sturdy steel frame to hold the camera modules in opposite and equidistant positions.
Screws (x50)	**$13.77**	
Washers (x50)	**$6.56**	
Nuts (x50)	**$4.65**	
Anker PowerLine Micro USB Cable (10ft) (x4)	**$40.00**	Used to power the microcontroller within a 10ft distance.
Anker 40W/8A 5-Port USB Charger PowerPort 5	**$29.37**	Power adapter to connect the four micro USB cables.
FINAL DESIGN COST: $578.83		

9.0 Recommendation

9.1 Object Detection Using Machine Learning

Object detection is one of the limiting factors in this project as they define the 3D space coordinate in which the object reside. The object detection process in this project is confined by a single colour space or a single ArUco 6X6 bits marker. This can be a problem when it comes to a system with multiple colour space and multiple ArUco markers.

Some of the solutions provided by open source software libraries such as TensorFlow and Darknet is to use machine learning algorithm to detect objects on an image. TensorFlow is a Google provided object detection API that identity object within an images. The machine learning model takes input image from the user and identify these objects while giving a percentage of confidence level. Similarly, another real-time object detection API is Darknet's YOLO. YOLO or "You Only Look Once", is a open source neural network written in C which trains a machine learning model that detects object given the input images.

9.2 Hardware Improvements

The hardware used in this project were chosen for its cost efficiency, programmability and availability. For instance, the Raspberry Pi and the Camera Module V2 are used as they are easily programmable and cost efficient. But through iterations of testing and data processing, it becomes apparent that the Pi and camera module has its limitations.

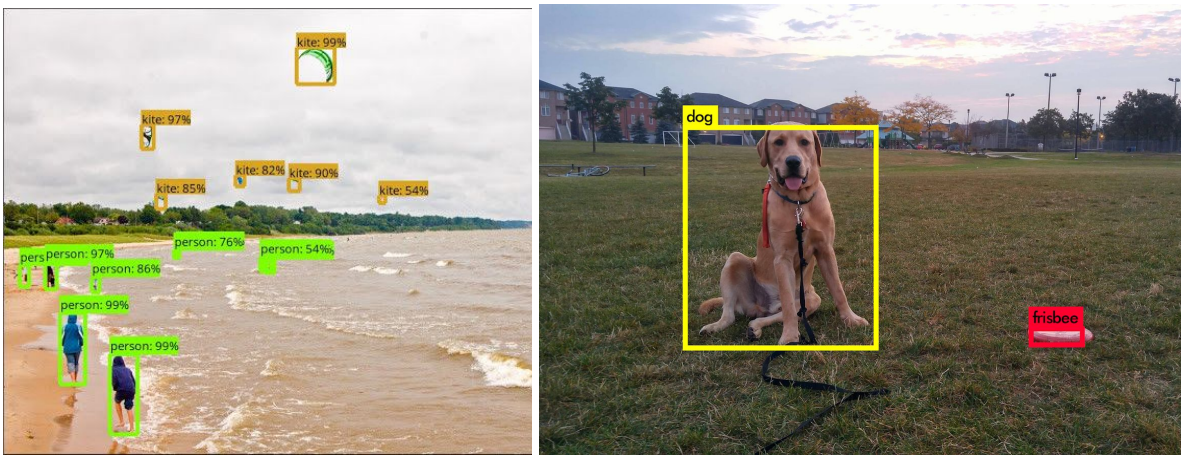


Figure 23. Object detection API: People flying kites, TensorFlow (left) and a dog with a frisbee, Darknet YOLO (right).

For instance, the camera module has a resolution limitation of 3280 x 2464 for still image and 1920 x 1080 at 30 FPS. This is an issue because the position of an object is dependent on the pixel resolution of and pixel accuracy. Higher resolution means better pinpoint location of the object's position. With the implementation of better cameras comes better resolution. The pixel accuracy enhances and output results from image resection and image intersection becomes better.

The Raspberry Pi is usually used for light computation projects. In the marker detection part of the project, a continuous function of the OpenCV software is needed to detect object and recognize the colour space or ArUco markers. This continuous function requires a lot of computation power, and a Pi is not suited to stream constantly at maximum resolution. This made the Pi overheat and shutdown. With a bigger budget, a better microcontroller invested to handle the streaming issues at higher resolution.

Another case that occurred during the data processing and data detection part of the project has to do with the Raspberry Pi wireless transfer of data. The Pi transfer at a maximum bit rate of 25 Mbps, this means it is impossible to transfer stream at 1080p and 30FPS through the Pi via WiFi. To get a faster bit rate, this means the Pi must avoid using wireless methods and instead use wire transfer of data. The wire transfer will have a limitation of the transfer speed of the USB 2.0 port (at 480 Mbps) built on to the Pi module.

9.3 Software Recommendations

The image resection and image intersection part of the iterative solution uses a written MATLAB code which derived from the collinearity equation. This can be tedious as it requires more runtime and takes up more resources as the code runs over a long period of time. To resolve this, the OpenCV provides a Camera Calibration function and 3D reconstruction function specifically its *solvePnP()*, and *calibrationMatrixValue()* methods. This function can be used to find the intrinsic parameter , image point, object point where the initial camera position (X_L,Y_L,Z_L)as well as object space coordinate (X,Z,Y) with respect to the camera system. To find the corresponding frame coordinate, a transformation must be performed with respect to the camera frame. This way, the function eliminates the need for

iterative process from calculating the resection and intersection, instead, a transformation process takes its place.

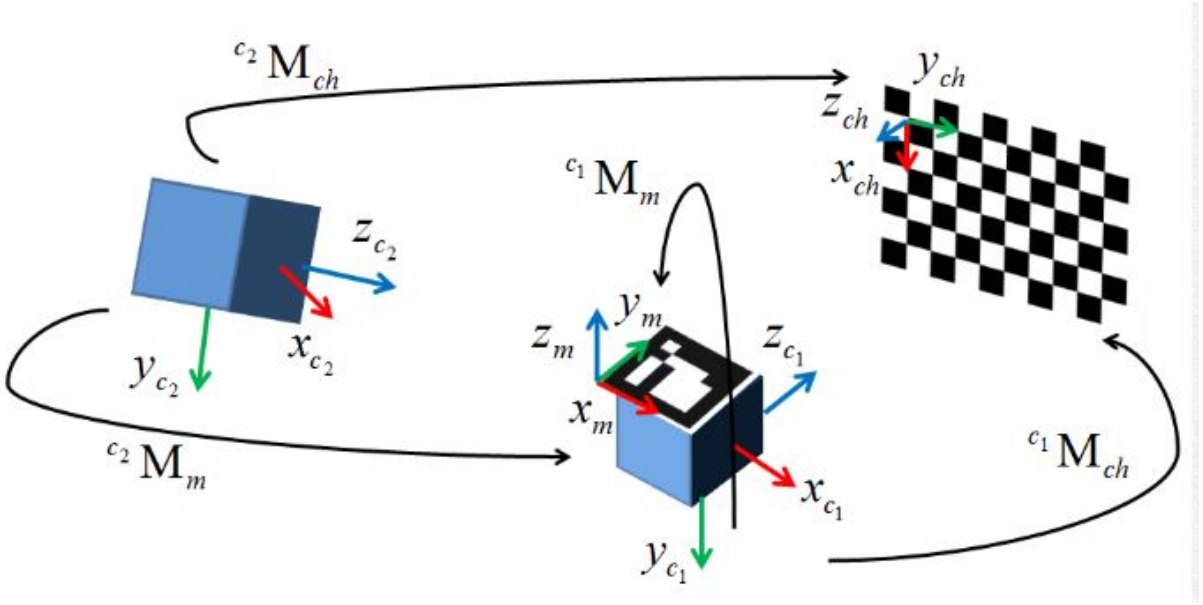


Figure 24. Camera Calibration and 3D reconstruction, solvePnP: describes the relationship between real-world coordinate (represented by the checkerboard and ArUco marker) to space coordinate in direction (X_{c2},Y_{c2}.Z_{c2}).

10.0 Conclusion

The proposed project was to incorporate the use of a local server to stream and process live video feed. The system that was delivered incorporates the use of a main controller that processes the live video feed to determine the position of an object with respect to the defined area. The delivered system provides a program, that is convenient for users to execute, and integrates the use of multiple components and other programs. The entire system was thoroughly tested to account for and satisfy all requirements stated in previous documents. Overall, the U-TRACKR system is able to carry out the specified functionality, and can be further expanded to control and coordinate the movements of multiple objects to widen the range of stakeholders.

10.1 Lessons Learned

In conclusion, our team polished the skills necessary for starting a career within the engineering industry. The team followed a variation of the waterfall development cycle and became familiar with the elements of the engineering design process which includes problem definition, specifications, background research, solution formulation, analysis, testing, and communication.

Furthermore, the team learned the basics in photogrammetry, space resection, and space intersection in order to implement the project. The team was able to apply the engineering knowledge, professional engineering practices, and project management skills to solve real-world problems. Most importantly, the team learned how to work efficiently together, resolve any conflicts during the course, and act with responsibility and competence as is necessary in a professional environment.

11.0 References

11.1 Literatures: Journals, Lectures, Textbook

[1]"Australia Zoo - Conservation", *Australiazoo.com*, 2018. [Online]. Available: http://www.australiazoo.com/conservation/projects/tracking/crocodiles/. [Accessed: 01- May- 2018].

[2]"Museum Trends: How Your Museum Can Collect and Use Data", *Locatify*, 2018. [Online]. Available: https://locatify.com/blog/museum-trends-how-your-museum-can-collect-and-use-data/. [Accessed: 01- May- 2018].

[3] K. Yoon, Y. Song and M. Jeon, "Multiple hypothesis tracking algorithm for multi-target multi-camera tracking with disjoint views", *IET Image Processing*, 2018.

[4] S. Bertrand, J. Marzat, M. Carton, C. Chaix, P. Varela, R. Waroux, G. De Ferron and R. Laurel, *A Low-Cost System for Indoor Motion Tracking of Unmanned Aerial Vehicles*, 2011.

[5] A. Harmat, M. Trentini and I. Sharf, "Multi-Camera Tracking and Mapping for Unmanned Aerial Vehicles in Unstructured Environments", *Journal of Intelligent & Robotic Systems*, vol. 78, no. 2, pp. 291-317, 2014.

[6] H. Oh, D. Won, S. Huh, D. Shim, M. Tahk and A. Tsourdos, "Indoor UAV Control Using Multi-Camera Visual Feedback", *Journal of Intelligent & Robotic Systems*, vol. 61, no. 1-4, pp. 57-84, 2010.

[7] C. Armenakis, "Analytical Photogrammetry Orthogonal Rotational Transformations", York University, Toronto, 2017.

[8] C. Armenakis, "Collinearity Equations and Applications", York University, Toronto, 2017.

[9] P. Wolf, B. Dewitt and B. Wilkinson, *Elements of Photogrammetry with Applications in GIS*,, 4th ed. McGraw-Hill Professional, 2014, pp. Chapter 10, 11.

11.2 Media: Images

[1] E. cv::SolvePnP, "Extrinsic Calibration With cv::SolvePnP", *Stackoverflow.com*, 2015. [Online]. Available: https://stackoverflow.com/questions/32851702/extrinsic-calibration-with-cvsolvepnp. [Accessed: 01- May- 2018].

[2]"Google releases new TensorFlow Object Detection API", *TechCrunch*, 2018. [Online]. Available: https://techcrunch.com/2017/06/16/object-detection-api/. [Accessed: 01- May- 2018].

[3] P. Wolf, B. Dewitt and B. Wilkinson, *Elements of Photogrammetry with Applications in GIS*,, 4th ed. McGraw-Hill Professional, 2014, pp. 11-5

11.3 API: MATLAB, Python, OpenCV

[1]"Camera Module - Raspberry Pi Documentation", *Raspberrypi.org*, 2018. [Online]. Available: https://www.raspberrypi.org/documentation/usage/camera/. [Accessed: 01- May- 2018].

[2]"Reading and Writing Images and Video — OpenCV 2.4.13.6 documentation", *Docs.opencv.org*, 2018. [Online]. Available: https://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html#videocapture-read. [Accessed: 01- May- 2018].

[3]"OpenCV: Detection of ArUco Markers", *Docs.opencv.org*, 2018. [Online]. Available: https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html. [Accessed: 01- May- 2018].

[4]"Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.6 documentation", *Docs.opencv.org*, 2018. [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html. [Accessed: 01- May- 2018].

[5]C. Calibrator and S. Calibrator, "Single Camera Calibrator App- MATLAB & Simulink", *Mathworks.com*, 2018. [Online]. Available: https://www.mathworks.com/help/vision/ug/single-camera-calibrator-app.html. [Accessed: 01- May- 2018].

12.0 Appendices

Appendix A: Space Resection

Table 19: Image Resection Input. Ground coordinates with preset space coordinates and measured pixel coordinates at a resolution of 3280X2464.

Ground Control Coordinates	**Space Coordinates** (X,Y,Z) [m]	**Pixel Coordinates** (x_{pixel},y_{pixel}) [pixels]
1	(0.45, 0.18, 0)	(2122, 761)
2	(0.63, 0.27, 0.061)	(1628, 641)
3	(0.72, 0.45, 0)	(1337, 849)
4	(0.63, 0.63, 0.061)	(1157, 1252)
5	(0.45 ,0.72, 0)	(1344, 1657)
6	(0.27, 0.63, 0.062)	(1648, 1854)
7	(0.18, 0.45, 0)	(2139, 1711)
8	(0.27, 0.27, 0.061)	(2240, 1238)
9	(0.72, 0.18, 0.146)	(1526, 263)
10	(0.72, 0.72, 0.145)	(806, 1260)
11	(0.18, 0.72, 0.146)	(1535, 2254)
12	(0.18, 0.18, 0.144)	(2554, 1237)
13	(0.45, 0.45, 0.146)	(1527, 1246)

Appendix B: Space Intersection

Table 20: Image Intersection Input. exterior orientation and Image coordinate of the object.

Cameras	**Exterior Orientations** (X,Y,Z,ω,ϕ,κ) [m, rads]	**Image coordinates** (x,y) [mm]
1	(0.0616, 0.0706, 1.0446, 0.3630, -0.35611, -2.3040)	(0.0756, -0.0039)
2	(0.7966, 0.0817, 1.0294, 0.3729, 0.35630, -0.8475)	(0.0174, -0.1182)
3	(0.7850, 0.7972 1.0223,-0.3631, 0.3145,-5.3319)	(-0.0902, -0.0106)
Intersection (X,Y,Z) [m]	(0.4190, 0.4115, 0.0915)	

13.0 Project Self-Evaluation

Table 21. Self-Evaluation

RUBRIC CRITERION FOR FINAL PROJECT DOCUMENTATION	SELF-EVALUATION RANKING	OUR JUSTIFICATIONS
Rubric 1: **Explain the importance of compliance with the Professional Engineers Acts and other relevant laws, regulations, intellectual property guidelines and contractual obligations and follow best practices**	**Level 3:** Adequately explains the importance of compliance and other relevant laws, regulations, intellectual property guidelines and contractual obligations and follow best practices	Section 3.3.3 goes over all regulatory requirements that were met during the timeline of this project, as well as the importance of each regulation.
Rubric 2: **Employ strategies for reflection, assessment and self-assessment of team goals and activities in multidisciplinary settings**	**Level 4:** Employs appropriate strategies for reflection, assessment and self-assessment of team goals and activities in multidisciplinary settings	Section 10.0 goes over the conclusions and lessons learned from the project, which includes a reflection on possible improvements.
Rubric 3: **Adhere to written instructions in a professional context**	**Level 4:** Completes tasks as instructed	The document goes over all major sections detailing the project, as indicated in section DID-2 of the governing document.
Rubric 4: **Evaluate critical information in reports and design documents**	**Level 4:** Evaluation of all critical information in reports and design documents; mostly to a professional standard	Sections 4.0 and 5.0 explain all the critical information and engineering design processes in the creation of this system.
Rubric 5: **Appraise possible improvements in the problem solving process**	**Level 4:** Evaluates possible improvements in the problem solving process	Section 9.0 explains all possible and realistic improvements to the system in both hardware and software aspects.
Rubric 6: **Justify the strength and limitations of the solution and make recommendation for possible improvements**	**Level 4:** Justifies the strengths and limitations of the solution; makes recommendations for possible improvements	Section 3.4 explains all the limitations, constraints, and improvements of the U-TRACKR system, and makes recommendations on possible improvements.