Technische Universität München

Institute for Flight System Dynamics

Mono Camera Indoor Position Tracking

–

Integrated Approach for UAV Flight Controller Testing

Master Thesis

Author: Florian Wachter

Matriculation Number: 3628475

Supervisor: Prof. Dr.-Ing. Florian Holzapfel

March 2017

Statutory Declaration

I, Florian Wachter, declare on oath towards the Institute of Flight System Dynamics of Technische Universität München, that I have prepared the present Semester thesis independently and with the aid of nothing but the resources listed in the bibliography. This thesis has neither as-is nor similarly been submitted to any other university.

Garching, March 18, 2017

Florian Wachter

Kurzfassung

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Table of Contents

List of Figures

List of Tables

Table of Acronyms

Acronym	Description
AC	Aerodynamic Center
AoA	Angle of Attack
CoG	Center of Gravity
DCM	Direction Cosine Matrix
DoF	Degrees of Freedom
FCC	Flight Control Computer
GA	Genetic Algorithm
GRM	Guided Research Missile
MCS	Monte Carlo Simulation
MoI	Moment of Inertia
PDF	Probability Density Function
WTD 91	Wehrtechnische Dienststelle 91
ZFW	Zero Fuel Weight

Table of Symbols

Latin Letters

Symbol	Unit	Description
g	m/s^2	Gravitational acceleration
p	rad/s	Roll rate
q	rad/s	Pitch rate
r	rad/s	Yaw rate
S	m^2	Reference area
d	m	Reference length
DCM	$-$	Rotation matrix from NED to body system
V_∞	m/s	Free airflow speed

Greek Letters

Symbol	Unit	Description
α	rad	Angle of Attack
β	rad	Sideslip Angle
Ψ	rad	Heading
Θ	rad	Elevation
Φ	rad	Bank

Indices

Symbol	Description
x	Variable related to axial force
y	Variable related to side force
z	Variable related to normal force
l	Variable related to roll moment
m	Variable related to pitch moment
n	Variable related to yaw moment
b	Variable given in the body frame

1 Introduction

2 Safety Information

3 Optical Mono Camera Position Tracking With *Rigid Track*

The now described position tracking system provides positions of objects on a sub-mm level. For that a minimum of 4 reflective markers have to be attached to the object. Those are then captured with one single camera and the frames are then processed on a connected computer. The software used for this is the open source computer vision library OpenCV and the GUI is created with Qt. Finally the resulting position and attitude is send at 100 Hz to a drone or a different device via UDP.

In the following sections the hardware is listed and explained. For a better understanding the algorithms employed in the software are theoretically explained and the source code is discussed. Then the system setup helps the user with guidelines how to mount the camera and connect everything. Before the tracking system can be used a calibration of camera and ground frame are necessary as described in section 3.5. And last but not least the operation, software and caveats to keep in mind close this chapter.

3.1 Hardware

The complete setup consists of following items:

1. 1 Optitrack Flex 3 camera
2. 1 3m USB 2.0 cable
3. 1 computer with Windows operating system
4. 4 reflective markers
5. 1 camera calibration pattern
6. 1 ground calibration frame
7. 1 tripod for camera mounting
8. 1 WiFi Stick or other network equipment for sending the results (optional)

3.1.1 Optitrack Flex 3 Camera

The main part of the setup is one Optitrack Flex 3 camera, from now on mentioned as Flex 3. Its applications are virtual reality, robotics, movement sciences and animation.

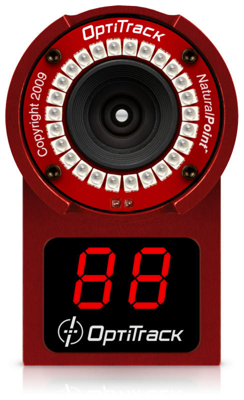


Figure 3-1: The Optitrack Flex 3 camera. The ring around the lens are the IR LEDs.(Source: `http://optitrack.com/about/mediakit/`**)**

The first difference when compared to normal consumer cameras is the higher frame rate of 100 Hz and the IR long pass filter. Normally IR deteriorates picture quality and hence most cameras have a IR blocking filter in front of the image sensor. In this application however the visible light is the disturbing factor and only the IR light is of interest. The camera is equipped with 26 IR LEDs that are placed around the Lens. This light is reflected by the retro reflective markers and sent back to the camera.

The second difference are the hardware implemented object detection algorithms. The camera does not have to send complete frames to the computer. Instead it detects markers with a simple brightness threshold and computes their properties like roundness, size and position in the camera image frames and then sends those parameters to the computer. This approach is faster than in software and saves bandwidth which is needed for applications where the maximum frame rate is more than 100 Hz or the image resolution is higher. (The Flex 3 is the most affordable camera OptiTrack offers. Better models achieve up to 360 Hz or a resolution of 2048 x 2048 pixels.) Internally those algorithms use grayscale images, that results in a higher precision on sub-pixel level [1], [2].

Camera Body	Weight 119 g Width 45.2 mm Height 74.7 mm Depth 36.6 mm Mounting 1/4 inch tripod thread
LED Ring	26 LEDs 850 nm IR Adjustable brightness
Lens	3.5 mm F#1.6 M12 Lens mount Field of view horizontal 58 deg, vertical 45 deg 800 nm IR long pass filter with filter switcher
Image Sensor	Resolution 640 x 480 Frame rate 25, 50, 100 frames per second Latency 10 ms Global shutter

Table 3-1: Flex 3 technical specifications
(Source: `http://optitrack.com/products/flex-3/specs.html`**)**

3.1.2 Reflective Markers

For robustness and faster processing retro reflective markers are attached to the object ought to be tracked. With this approach there is no need to run advanced object recognition algorithms on the computer since a simple brightness threshold algorithm is sufficient. Every pixel thats brighter than the specified threshold is assumed to be part of a marker. Further those pixels are merged together if they are adjacent. For indoor purposes normal sizes markers are used, if the distance between Flex 3 and the markers is larger, bigger markers can be used. For difficult environments with a lot of normal light IR LEDs can be used instead or a better camera has to be purchased. OptiTracks Prime 13 camera e.g. has more IR LEDs that can operate in a brighter strobe mode that is enough even for outdoor use.

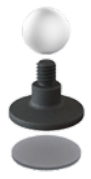


Figure 3-2: The reflective markers that are tracked by the camera.(Source: `http://optitrack.com/parts-accessories/`**)**

3.1.3 Camera Calibration Pattern

Pictures of the real world taken by a camera are always distorted. For accurate tracking results the distortion coefficients have to be known and fed into the subsequent algorithms. For that purpose a chessboard pattern has to be printed out with a know size and mounted on a flat plane, see Figure 3-3. The real world dimension of one square must be known in mm.

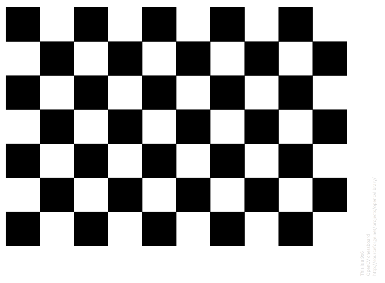


Figure 3-3: Camera calibration chessboard pattern for computing the distortion coefficients.(Source: `http://docs.opencv.org/2.4/_downloads/pattern.png`**)**

3.1.4 Ground Calibration Frame

The Flex 3s position relative to the ground could be computed with the object to be tracked standing on the ground (with attached markers). But depending on the structural stiffness of the object and the accuracy the markers were placed, the obtained values for position and attitude are more or less precise. Hence badly placed markers etc. can lead to a biased and skewed ground reference system.

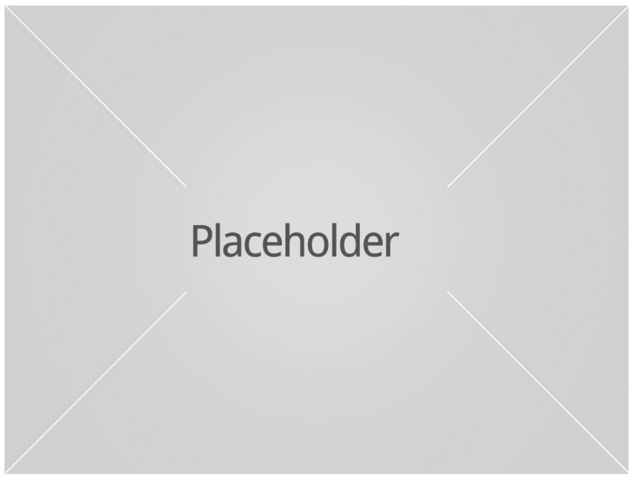


Figure 3-4: Ground calibration frame with attached markers

3.2 Theoretics of Pose Estimation

To obtain the pose of an object, that means its position and attitude, hence all 6 degrees of freedom, is a common problem in computer vision. Luckily many algorithm exist that tackle the problem. While some require a stereo camera setup it is also possible to calculate the pose with only one monocular camera. In the literature this problem is called Perspective-n-Point problem. From our daily observations it is evident that one camera is sufficient for this task. To assist the user and show up potential limitations and caveats when using this software the theory behind is briefly discussed.

3.2.1 3D Projection

The model used for projection of 3D points into 2D is that of a pinhole camera. Projection of a point in three dimensional space onto an image created by a camera can be expressed as a matrix-vector multiplication as shown in Equation 3.2.1.

$$s \underbrace{\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}}_{m'} = P^{3\times 4} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{M'} \tag{3-1}$$

The vector M' are the 3D coordinates of a given point in a world CoSy. For our purposes it is equal to the marker CoSy. In computer vision the 1 as 4th element is often added to obtained so-called homogeneous coordinates that enable merging of rotation and translation transformations. The matrix P is the linear projection matrix or camera matrix. Multiplication of M' and P yield the coordinates of the 3D point in pixel coordinates as seen on the image the camera provides. The respective camera image CoSy has its origin in the upper-left corner of the image.

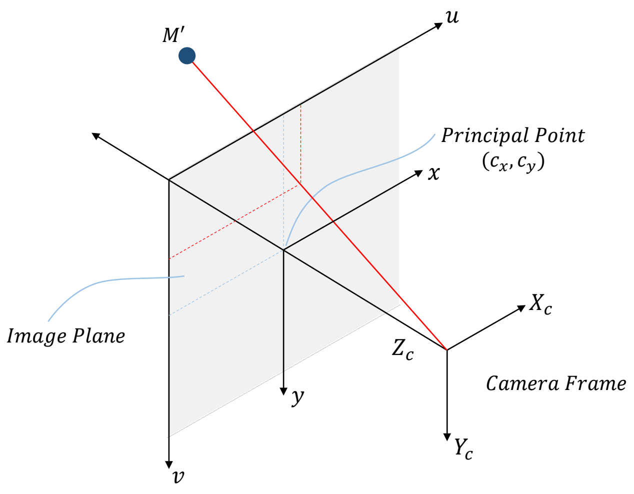


Figure 3-5: The pinhole camera model projects 3D points onto 2D

The next step is the determination of P, it can be split up as a product of two simpler matrices $K \times [R|t]$. Former is called calibration matrix or intrinsic parameters and latter is named extrinsic parameters. The naming becomes clearer when we take a look at their entries.

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \qquad [R|t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{3-2}$$

Intrinsic Parameters All parameters in the K matrix only depend on the camera itself, not on its orientation or position. The two f-values denote the focal length and reflects the field of view. A smaller f results in a bigger FoV and vice versa. As the camera image does not have to be quadratic (The Flex 3 has a resolution of 640x480 pixels with equally sized pixel in each direction) the horizontal and vertical FoV can differ as the f_x and f_y do.

As in Equation 3.2.1 depicted a M' that lies on the Z_c axis has non-zero values for u and v, instead their value is c_x and c_y. This point is named principal point and denotes the translation from camera image CoSy and camera CoSy.

The last parameter s determines skewing of the picture but for most cameras it takes on zero as value.

But how to obtain those parameters? Despite the f-values can be computed with the FoV or camera data sheet those nominal values differ from the real camera intrinsics. And also the c-values may not coincide with the respective camera resolution divided by two. This problem can be tackled by camera calibration which computes those parameters for this specific camera, see section 3.5.

Extrinsic Parameters Since the calibration matrix projects 3D points given in the camera CoSy onto the image plane, but M' is given in the arbitrary world CoSy we, need a CoSy transformation. That is done by multiplying M' with $[R|t]$, which is composed of a rotation matrix and a translation vector. We also see that the use of homogeneous coordinates makes it possible to write rotation and translation in one matrix instead of calculating $R\times(XYZ)^{\intercal}+\vec{t}$. But one problem arises, how to obtain $[R|t]$? This is itself the problem of pose estimation, here the pose of the camera CoSy relative to the world CoSy. Additionally, K can only be computed if $[R|t]$ is know. In practice camera calibration finds solutions for both K and $[R|t]$ in the same step, calculating only one of them is hardly possible.

Camera Distortion Until now the model of an ideal pinhole camera was assumed. In reality however pictures taken are distorted, especially with cheap cameras or big FoVs (hence small focal lengths) this has to kept in mind. Mathematically this can be accounted for by distortion factors that are split up in radial and decentering distortion with the first one having a bigger effect than latter. A reasonable model for both is shown in Equation 3.2.1 where $\check{x}$ is the distorted pixel position of the undistored pixel position x.

$$\check{x} = x + \delta_x \quad \check{y} = y + \delta_y \tag{3-3}$$

$$\delta_x = x(k_1r^2 + k_2r^4 + k_3r^6 + \dots) + [p_1(r^2 + 2x^2) + 2p_2xy](1 + p_3r^2 + \dots) \tag{3-4}$$

$$\delta_y = \underbrace{y(k_1r^2 + k_2r^4 + k_3r^6 + \dots)}_{radial} + \underbrace{[2p_1xy + p_2(r^2 + 2y^2)](1 + p_3r^2 + \dots)}_{decentering} \tag{3-5}$$

$$r = \sqrt{x^2 + y^2} \tag{3-6}$$

For practical use only the biggest (first few) parameters are estimated during camera calibration. Both effects compared, radial distortion affects image capturing much more than the decentering one. In OpenCV only k_1, k_2, k_3, p_1 and p_3 are considered[1].

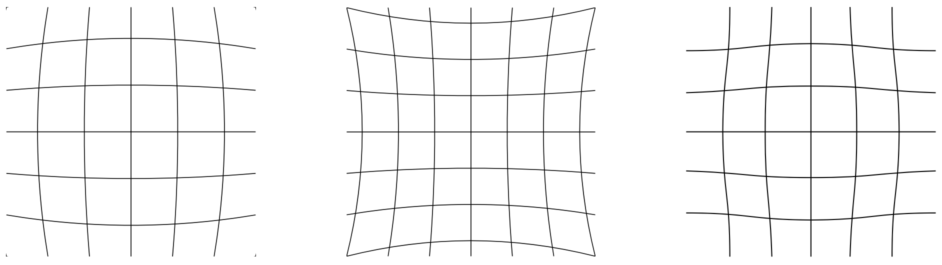


Figure 3-6: Radial distortion, on the left barrel distortion, in the middle pincushion distortion and right mustache distortion. (Source:`https://en.wikipedia.org/wiki/Distortion_(optics)`**)**

For the Flex 3 the intrinsic and distortion parameters are computed with OpenCV and listed in Equation 3.2.1.

$$[k_1, k_2, p_1, p_2, k_3] = [-1.14 \times 10^{-1}, 9.13 \times 10^{-2}, 9.63 \times 10^{-4}, 4.46 \times 10^{-4}, 4.713 \times 10^{-2}] \tag{3-7}$$

$$K = \begin{bmatrix} 590 & 0 & 307 \\ 0 & 588 & 250 \\ 0 & 0 & 1 \end{bmatrix} \tag{3-8}$$

[1]`http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html`

3.2.2 Perspective-n-Points

When the camera is calibrated, so intrinsic and distortion parameters are known, the last step is the computation of $[R|t]$ which is the pose estimation problem. In computer vision this problem is called Perspective-n-Points and tries to obtain the pose with n given 3D points and their respective 2D projection.

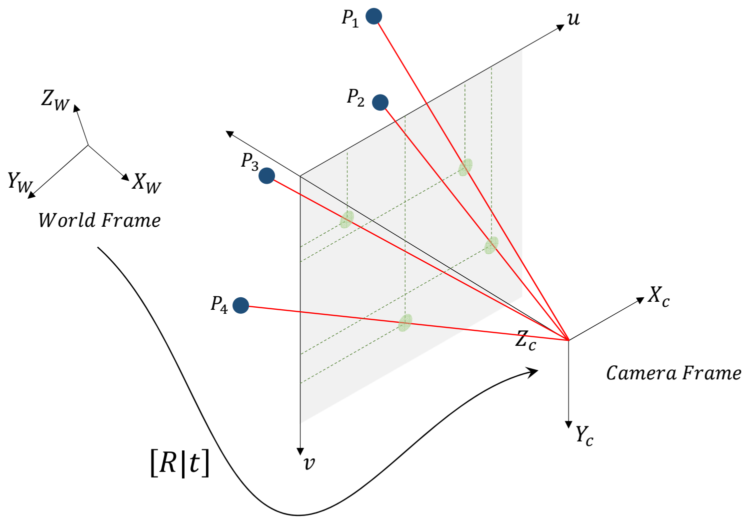


Figure 3-7: The Perspective-n-Points algorithm computes $[R|t]$ from the 3D world points P_i in blue and their corresponding 2D projections depicted by the green points.

The authors of [3] provided a solution and algorithm for the special case with 4 points. Despite its name "P3P" it needs 4 point correspondences to work, with only 3 points the results are ambiguous. Other algorithms like EPnP by [4] and iterative ones work with 4 or more point correspondences. Nevertheless OpenCV provides "P3P", "EPnP" and "Iterative".

Calling OpenCVs solvePnP function looks as following:

```
solvePnP(list_points3d, list_points2d, cameraMatrix, distCoeffs,
                                        Rvec, Tvec, useGuess, methodPNP);
```

Every argument of the function call is mandatory. A deeper insight provides the following table:

In practice it was found that *P3P* is more robust and faster regarding imprecise marker position measurements but *Iterative* is more precise under normal conditions. *EPnP* never yielded better results nor it was superior in robustness. For the Flex 3 and its refresh rate of 100 Hz the performance difference plays no role. Nevertheless for convenience it is possible to switch between those different methods in the GUI even while tracking.

⚠ CAUTION Switching between solvePnP methods can result in different values for Rvec and Tvec in a non-negligible magnitude, especially when marker positions are not exactly determined. It is recommended to try the different methods and chose the most appropriate one prior clicking *Set Zero* and *Start Tracking*.

As stated before solvePnP provides the rotation marker system to from camera CoSy in axis-angle notation, hence a 3d vector showing in rotation axis direction and the length of it denoting the rotation angle θ. For subsequent use OpenCVs function Rodrigues is applied on the rotation vector to obtain a rotation matrix. Finally the rotation matrix is decomposed into Euler-Angle notation. Mathematically following equations are computed.

Parameter	Type	Input/Output	Description
list_points3d	cv::Mat float	Input	The position of every marker in the marker coordinate system, depicted as blue points in Figure 3-7. The unit can be meter or millimeter, in *Rigid Track* mm are used.
list_points2d	cv::Mat float	Input	The position of every marker in the camera picture frame, represented by the green points in 3-7. Units are in pixels, but decimal values obtained by the sub-pixel-accurate marker detection algorithm can be use for enhanced precision.
cameraMatrix	cv::Mat float	Input	The camera matrix as obtained by the calibration.
distCoeffs	cv::Mat float	Input	Distortion coefficients as obtained by the calibration, used to de-distort the captured pictures.
Rvec	cv::Mat float	I/O	The rotation from marker system to camera coordinate system in axis-angle notation. If useGuess is true, the value of Rvec is used as starting value for the solvePnP algorithm.
Tvec	cv::Mat float	I/O	The translation vector from the camera coordinate system to the marker system given in the camera CoSy. If useGuess is true, the value of Tvec is used as starting value for the solvePnP algorithm.
useGuess	bool	Input	Iterative methods for solvePnP need starting values for both Rvec and Tvec. Dependent on those values the convergence may be faster or slower. If useGuess is true, the Rvec and Tvec arguments are used for initial values.
methodPnP	int	Input	OpenCV offers different solution algorithms for the P-n-P problem. The options are *Iterative*, *P3P* and *EPnP*. The other two options *DLS* and *uPnP* are not correctly implemented in OpenCV and *EPnP* is used as fallback.

Table 3-2: SolvePnP arguments as stated in the OpenCV documentation(Source: `http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`**)**

$$Rvec_{CM} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \tag{3-9}$$

$$\theta \leftarrow norm(Rvec_{CM}) \tag{3-10}$$

$$r \leftarrow Rvec_{CM}/\theta \tag{3-11}$$

$$R_{CM} = \cos\theta I + (1-\cos\theta)rr^{\mathsf{T}} + \sin\theta \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \tag{3-12}$$

$$\begin{bmatrix} \Psi \\ \Theta \\ \Phi \end{bmatrix} = decomposeProjectionMatrix(R_{CM}) \tag{3-13}$$

The result R_{CM} of Equation **3-12** is the rotation Matrix from marker system to camera coordinate system. For better understanding this rotation matrix is decomposed with OpenCVs function *decomposeProjection-Matrix* which takes a projection matrix as input and returns camera matrix, rotation matrices and Euler angles [5].

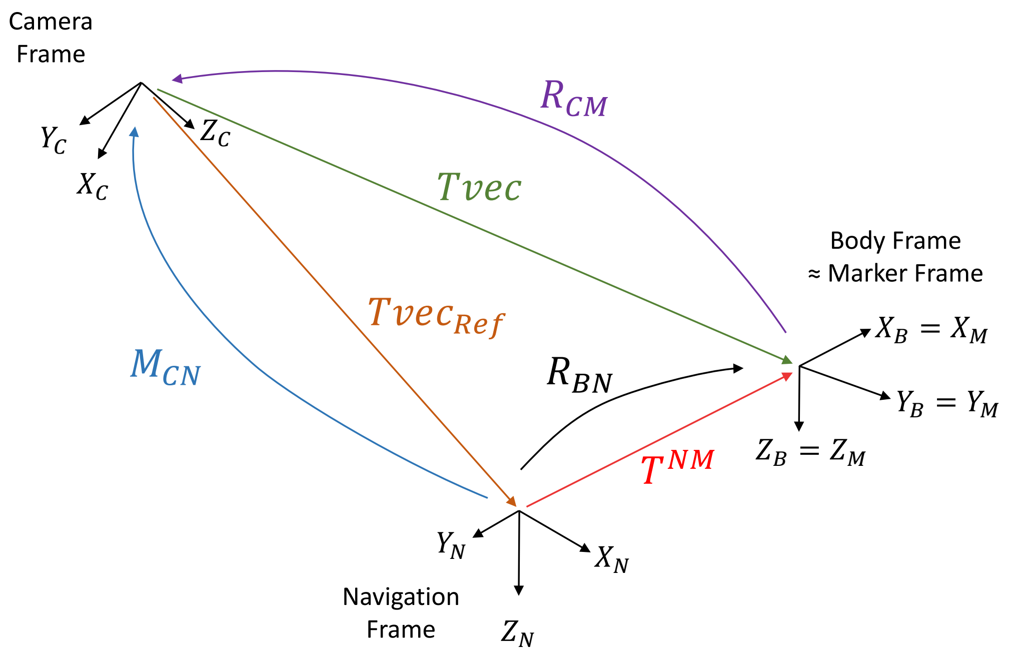


Figure 3-8: The transformations between camera, navigation and body/marker frame.

To obtain the rotation and translation in the navigation frame of the tracked object a few more transformations are underdone.

$$T_C^{NM} = Tvec - Tvec_{Ref} \tag{3-14}$$

$$T_N^{NM} = M_{NC} \times T_C^{NM} \tag{3-15}$$

$$R_{NM} = M_{NC} \times R_{CM} \tag{3-16}$$

3.3 Tracking Software *Rigid Track*

The present software was developed that enables optical position tracking in combination with an OptiFlex camera. The programming language was Visual C++ 2015, the computer vision library OpenCV (Version

3.2 in x64) used for the pose-estimation algorithms. To communicate and control the cameras and lever their hardware accelerated marker detection the OptiTrack Camera SDK library (CameraLibrary2013x64S.lib) is included. The GUI is powered by the widespread Qt framework (Qt 5.7.0) and also delivers classes for network communication and other co-routines like file handling.

3.4 Setup

Before tracking can be used both software *Rigid Track* and hardware have to be set up. The prerequisites for *Rigid Track* are:

- A x64 capable PC with Windows 10. While it should also work on Windows 8 it was only tested with the latest version. The OS has to be x64 as well. Minimum requirements regarding processor performance or memory are not determined, any recent computer that is capable of Windows 10 should be sufficient.
- Visual C++ Redistributable für Visual Studio 2015[2]
- *Rigid Track* Installation Program

3.4.1 Rigid Track Installation

Start the RigidTrack_setup.exe and follow the instructions given in the installation assistant. Default parameters like installation directory or shortcuts to be created can be chosen. But normally clicking *continue* and keeping the default values should be sufficient. When the installation is completed a shortcut in the start menu and the desktop can be used to start *Rigid Track*.

Figure 3-9: *Rigid Track* shortcut created in the start menu and the desktop.

3.4.2 Rigid Track User Interface

Figure 3-10 shows the *Rigid Track* user interface after starting the program. The specific functions can be started directly by clicking on the user interface elements. A click on *Help* opens this help document.

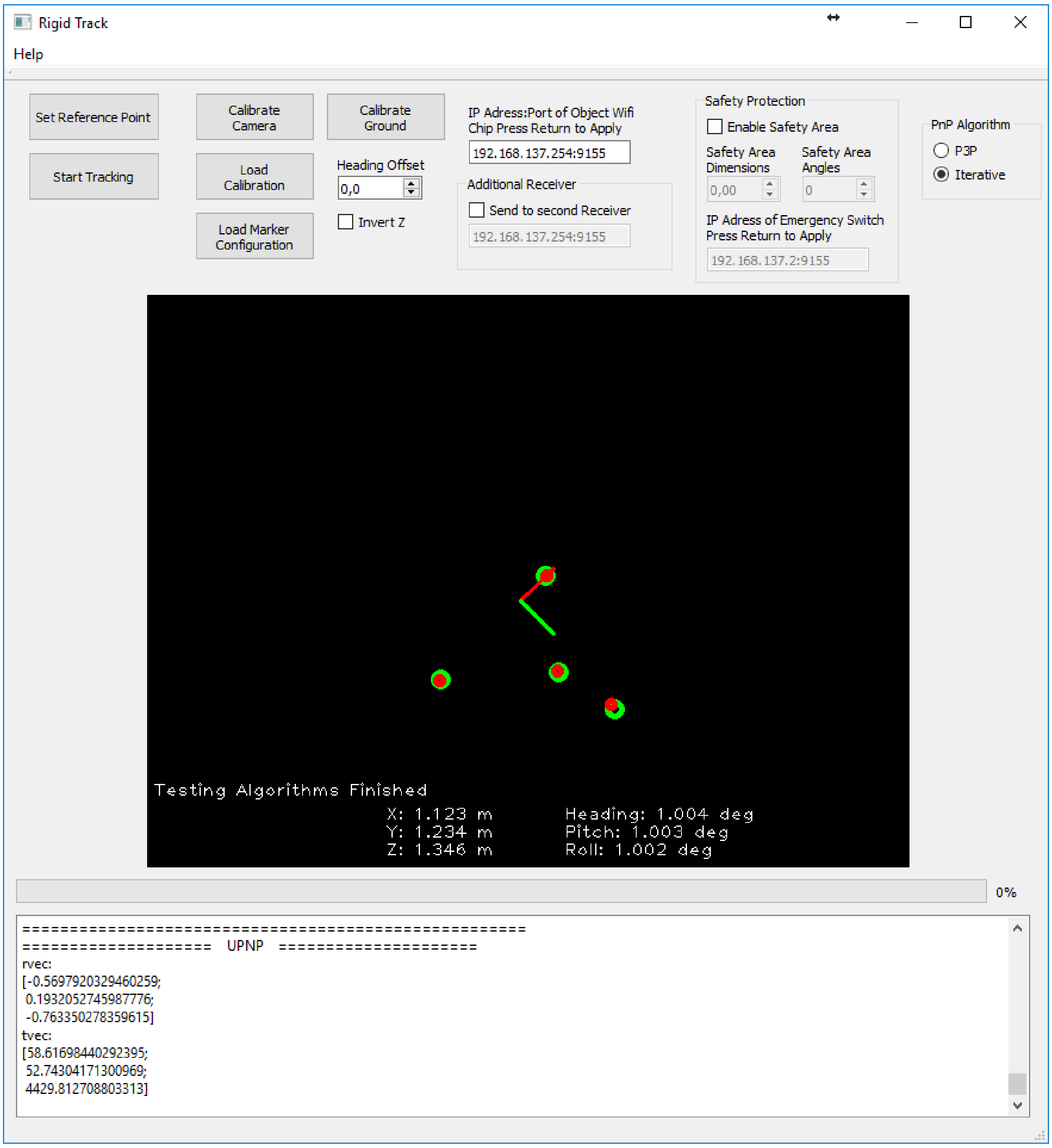
A comprehensive explanation of each element can be found in section 3.6.

3.4.3 Hardware Installation

Complete the following steps to successfully set up the camera. A picture of the hardware setup is shown in Figure 3-11.

1. Mount the Flex 3 on a tripod or any other stable object. Use the 1/4 inch tripod thread for that. After calibration is must not be moved or tilted in any direction.
2. Connect the provided USB 2.0 cable to te Flex 3 and the computer that is running *Rigid Track*.

[2]`https://www.microsoft.com/de-de/download/details.aspx?id=48145`



Heading Offset: 0,0
Invert Z: [ ]
IP Adress:Port of Object Wifi Chip Press Return to Apply: 192.168.137.254:9155
Send to second Receiver: [ ] 192.168.137.254:9155
Enable Safety Area: [ ]
Safety Area Dimensions: 0,00
Safety Area Angles: 0
IP Adress of Emergency Switch Press Return to Apply: 192.168.137.2:9155
PnP Algorithm: P3P [✓ Iterative]

```
Testing Algorithms Finished
                    X: 1.123 m        Heading: 1.004 deg
                    Y: 1.234 m        Pitch: 1.003 deg
                    Z: 1.346 m        Roll: 1.002 deg
```

```
=======================================================
====================  UPNP  =====================
rvec:
[-0.5697920329460259;
 0.1932052745987776;
 -0.763350278359615]
tvec:
[58.61698440292395;
 52.74304171300969;
 4429.812708803313]
```

Figure 3-10: *Rigid Track* user interface.

3. Mount at least 4 markers on the object that should be tracked. With exactly 4 markers every algorithm supported by *Rigid Track* can be chosen. With more markers the P3P algorithm can not be chosen and instead *Iterative* or *EPnP* have to be used.

4. Specify the Cartesian coordinate system, also called marker system, of the object to be tracked. The position and attitude computed by *Rigid Track* always reference on that system. There are no special requirements on the location or orientation of this system.

5. Measure the coordinates of every marker regarding the previously specified coordinate system in millimeters. Those values are later needed during operation, see section 3.6.

6. Navigate to the install directory of *Rigid Track* and create a copy the file*marker_standard.xml*.
7. Open the copy and change the existing values to the previously in step 5 measured marker positions.
8. Save the file under a name of choice. If the configuration should be loaded at startup of *Rigid Track*, save it as *markerStandard.xml* overwriting the old one.

NOTICE Useful tips that help choosing the right place with minimal environmental influences are stated on OptiTracks documentation website, see [6] fo further information. When *Rigid Track* can not find the right amount of markers start the executable *visualtest.exe* that resides in the installation folder or via the *Rigid Track* user interface. This program shows the camera view and allows the user to change some parameters as exposure. Check if any reflections other by markers are visible. If yes cover objects creating those by something non reflective. In practice crepe showed excellent results.

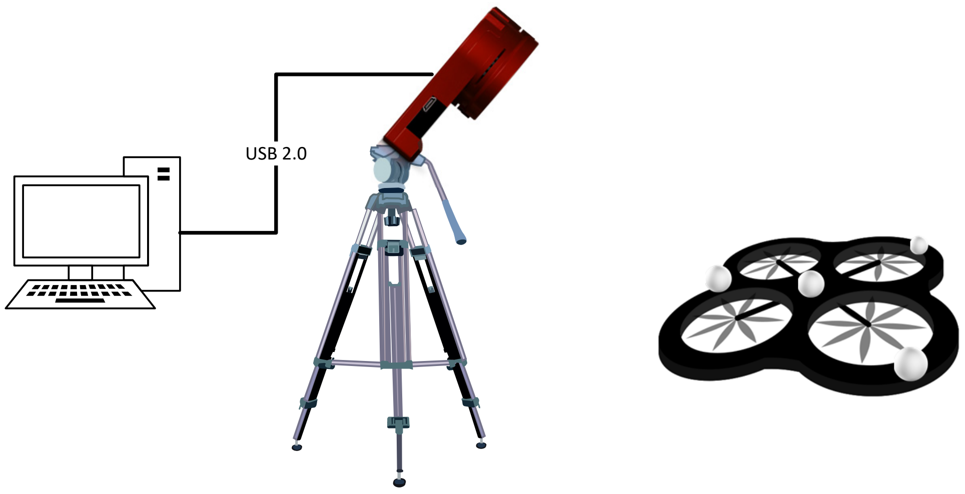


Figure 3-11: Setup arrangement and connections. At least 4 markers have to be attached on the object.

3.5 Calibration

Prior first use the Flex 3 and its pose must be calibrated to yield accurate results. While the camera calibration itself only has to be done once, regardless its pose, the pose calibration itself must be run every time the camera is moved.

3.5.1 Camera Calibration

As explained in paragraph 3.2.1 the picture taken by any camera is distorted. With the help of a distortion model one can try to reduce this negative influence by applying the inverse distortion. *Rigid Track* uses a 5 parameter model, see Equation 3.2.1, as provided by OpenCV[3]. Computation of these 5 parameters have to be done prior tracking to achieve precise results. Same counts for the camera matrix, focal length and principal points differ between cameras and depend on other things as like focus and lens center offset.

To calibrate the camera and save the camera matrix and distortion coefficients for later use do the following steps:

[3]http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

1. Print the checkerboard pattern (subsection 3.1.3) on paper and glue it on a rigid sheet of cardboard or similar. For some cameras it is sufficient to display the pattern on a monitor but Flex 3's IR Filter blocks the image shown on a monitor.
2. Start *Rigid Track*.
3. Click on the *Calibrate Camera* button. This starts the calibration routine.
4. Move the Flex 3 by hand and try to capture images of the checkerboard pattern. Alternatively keep the Flex 3 mounted as it is and move the calibration pattern around. The software recognizes it and computes the distortion parameters. When 80 frames with patterns are sampled the camera matrix, distortion coefficients and RMS of the calibration are shown in the log.
5. When prompted to save the calibration choose a directory and file name of choice and press *Save*.
6. To set this file as standard that is applied at program startup, name it *calibration.xml* and save it in the rigidtrack.exe directory. The next time *Rigid Track* is started the calibration file is loaded automatically.

⚠ CAUTION Touching the camera lens, focusing the camera manually or enabling the IR filter alter the camera matrix and deteriorate the tracking accuracy. In this case recalibrate the camera. If the IR filter is used during tracking make sure it is also enabled during calibration.

⚠ CAUTION Badly mounted cameras or mounting on flexible or vibrating structures deteriorates the tracking results. To prevent this attach the camera firmly on a rigid object. Avoid vibrations or other influences by persons walking by the camera.

3.5.2 Camera Pose Estimation

Since the rotation from camera to navigation coordinate system must be known, see Equation **3-12**, a camera pose estimation step is necessary every time the camera is moved with respect to the ground. Also the translation from camera to navigation coordinate system is step of this process. But when needed, this translation vector can be exchanged during the *Set Zero* routine. To obtain and save M_{NC} and $Tvec_{Ref}$ do the following steps.

1. Place the ground calibration frame on the ground such that its origin and attitude correspond with the navigation coordinate system that is intended to be used.
2. Start *Rigid Track*.
3. The marker positions of the calibration frame are already saved in the file *calibrationFrame.xml*. Click on *Load Marker Configuration* and select this file.
4. Click on *Calibrate Ground*.
5. The routine now averages 200 poses of the ground calibration frame. After that the rotation matrix M_{CN} and the position $Tvec_{Ref}$ are saved in the *RigidTrack.exe* directory specified by the pup-up save dialog.
6. If this calibration data should be used as standard save it as *referenceData.xml*. The next time *Rigid-Track* is started this is automatically loaded.

3.5.3 SolvePnP Precision and Delay

For the calibrated Flex 3 Table 3-3 shows measurement standard deviations, hence the precision of *solvePnP*. The method was *Iterative* and the ground calibration frame was the object to be tracked. During the measurement this frame was not moved. The measured values follow a normal distribution. Due to the numerical properties of the algorithm and P-n-P problem itself the precision is different for specific directions. The best values for position precision are achieved in the plane parallel to the camera sensor. For Table 3-3 the Z_C axis was facing down approximately 24 $\deg$ and the distance to the object was 5 m. As consequence the standard deviation in X_O and Y_O is better than in Z_O direction. The same holds for angles, also the one that is in the plane parallel to the camera sensor is most precise. In this example the standard deviation for roll and pitch are magnitudes bigger than for the heading. The same principles can be applied to velocities since they are computed with finite differences. The time difference between two consecutive frames can probably be neglected for most use cases.

Time between the objects movement and the output by *Rigid Track* is derived by addition of different time delays. First part is the delay in the camera electronics, for the Flex 3 it is 10 ms. After that the software has to compute the pose which adds approximately 1 ms. The last part is sending the data to other computers. With Ethernet 1 ms are added and over WiFi 1 ms to 40 ms, depending on direct connection or routed. In total a delay of around 12 ms to 50 ms, depending on the network architecture, can be assumed.

Parameter	Standard Deviation
Position X in Navigation Frame X_O	0.118 mm
Position Y in Navigation Frame Y_O	0.051 mm
Position Z in Navigation Frame Z_O	0.156 mm
Velocity X in Navigation Frame	0.009 337 m s^{-1}
Velocity Y in Navigation Frame	0.006 671 m s^{-1}
Velocity Z in Navigation Frame	0.020 443 m s^{-1}
Roll	0.011 074 $\deg$
Pitch	0.011 513 $\deg$
Heading	0.003 154 $\deg$
Time Difference	0.151 ms

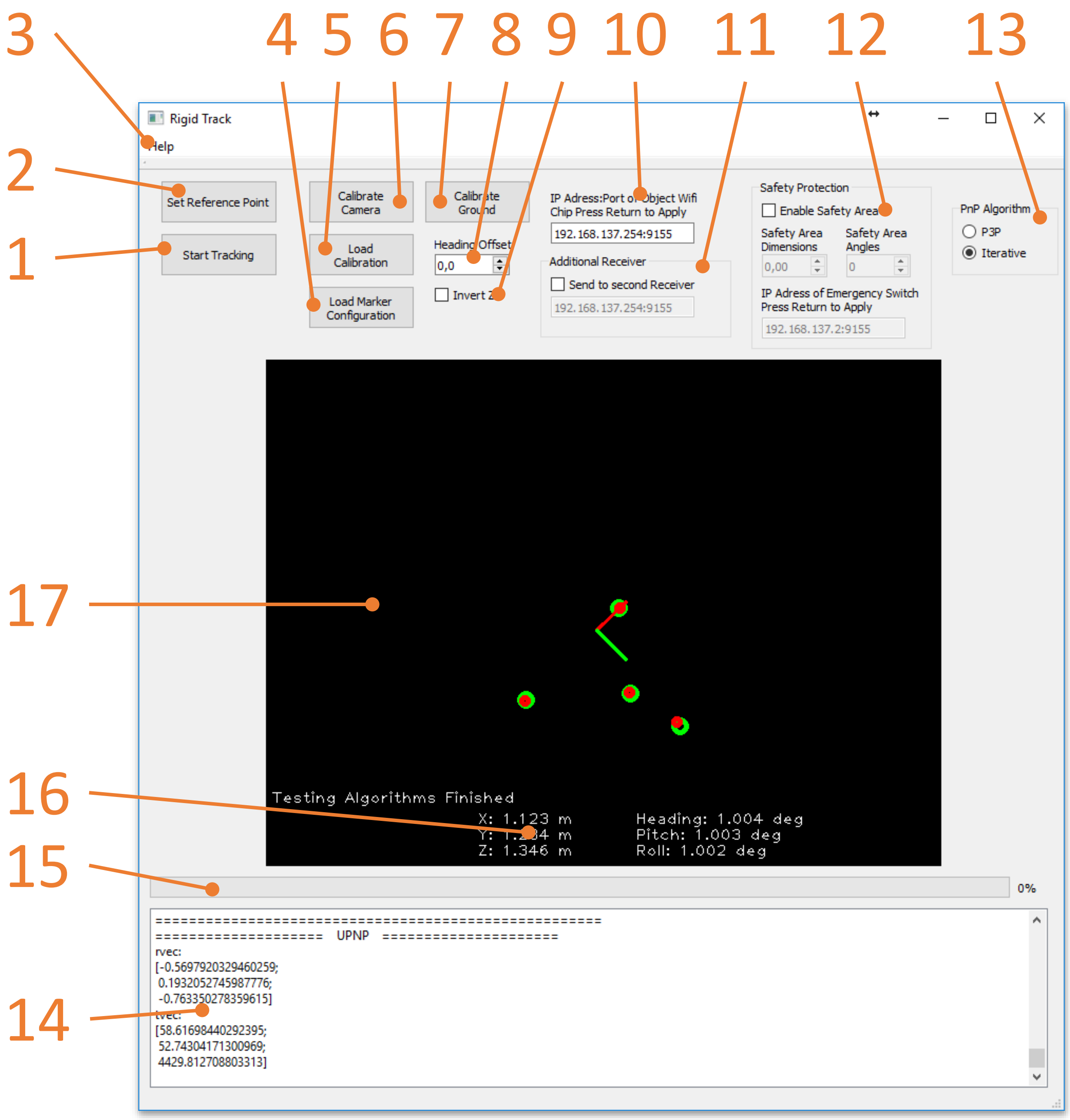
Table 3-3: SolvePnP precision with *Iterative*-method. Tracked object was the non-moving ground calibration frame.

NOTICE

Another parameter influencing the precision is the distance between markers. If they are wide apart the precision increases. But on the other hand the maximum tracking volume decreases as the markers exit the cameras FoV earlier. Instead of bigger marker distances the camera can be moved closer to the tracked object, but this results in the same trade off. This principle can be visualized by focusing an object with the eye, the nearer one is standing to it the smaller movements can be distinguished. But as the object moves it might exit the field of view. In practice keep the tracking volume as small as possible and as big as needed.

3.6 Operation and User Interface

1. Start Tracking Starts the tracking until the button is pressed again. When started the text of this button switches to *Stop Tracking*. As with *Set Zero* the camera exposure is computed beforehand. During tracking the coordinates and Euler angles are displayed in the bottom middle of the *camera view*. Simultaneously following values are saved in a comma-separated log file named *positonLog_DD_MM_YYYY_HH_MM_SS.txt*



Heading Offset: 0,0

IP Adress:Port of Object Wifi Chip: 192.168.137.254:9155

Send to second Receiver: 192.168.137.254:9155

Safety Area Dimensions: 0,00 — Safety Area Angles: 0

IP Adress of Emergency Switch: 192.168.137.2:9155

PnP Algorithm: P3P [✓ Iterative]

```
Testing Algorithms Finished
                 X: 1.123 m        Heading: 1.004 deg
                 Y: 1.2[illegible]4 m        Pitch: 1.003 deg
                 Z: 1.346 m        Roll: 1.002 deg
```

```
======================================================
====================  UPNP  =====================
rvec:
[-0.5697920329460259;
 0.1932052745987776;
 -0.763350278359615]
tvec:
[58.61698440292395;
 52.74304171300969;
 4429.812708803313]
```

Figure 3-12: *Rigid Track* user interface.

in the folder *logs*: hardware times stamp delivered by the Flex 3, position in X_N, Y_N and Z_N direction in [m], Euler angles Ψ, Θ and Φ in [deg] and velocity in X_N, Y_N and Z_N direction, [m/s]. Additionally position and Euler angles are sent via UDP to the previously defined IP adress/port combination (see point (10) and (11) with 100 Hz.

2. Set Reference Point Click on this button to set the position and attitude at which all values are zero. The pose is calculated 200 times and averaged. From now on every deviation from this reference pose are printed in the camera view (17), sent to other computers via UDP and saved in the log file. Prior all this the camera exposure is set by finding the minimum and maximum exposure where all marker points are visible. Then the exposure is set to the mean value of both.

3. Help Opens this document for help and information.

4. Load Marker Configuration When tracking different objects with different marker placements it is convenient to save the marker configuration for each and load it when needed. After choosing this option a dialog opens and the user has to select a previously created marker configuration file. After a click on *Open* the message log (14) confirms the changes and shows values for the configuration file name, number of markers and marker positions. If the number of markers is not 4 a notice is added to the message log that P3P is disabled since it only works with 4 markers. Additionally the P3P option in (13) is disabled.

5. Load Calibration Loads a previously saved camera calibration file. It contains the camera matrix and distortion coefficients. The message log (14) displays the file name loaded, the camera matrix and distortion coefficients.

6. Calibrate Camera This starts the camera calibration routine, refer to section 3.5 for the procedure.

7. Calibrate Ground A click on *Calibrate Ground* starts the camera pose calibration routine, follow the guidelines in subsection 3.5.2.

8. Heading Offset

9. Invert Z

10. IP Address and Port for receiver

11. Enable second receiver

12. Enable Safety Area

13. PnP Algorithm

14. Message Log

15. Progress Bar

16. Real Time Tracking Data

17. Camera View In this field the camera frame is shown in realtime. Tracking markers are shown as gray points. Red dots show the projection of the marker coordinates onto the camera image with $Rvec$ and $Tvec$ as computed by *solvePnP*. Under good conditions the red dots overlay the gray markers. Hence poor precision due to bad calibration or imprecise marker placement results in both, gray points and red dots, visibility. Additional information is overlaid by the Flex 3.

3.7 Troubleshooting

3.8 Source Code

3.8.1 Files and Functions

Following table describes the files in the *Rigid Track* project folder.

File Name	Description
main.cpp	C++ source file that contains non-GUI related functions. They are listed in Listing 3.1.
main.h	C++ header file for main.cpp.
RigidTrack.cpp	C++ source file that handles the GUI related functions.
RigidTrack.h	C++ header file for RigidTrack.cpp.
communication.cpp	C++ source file that implements a commObject class for interaction between main.cpp and RigidTrack.cpp.
communication.h	C++ header file for the commObject class.
RigidTrack.ui	GUI layout file.
libEGL.dll	QT Framework 4.7 libraries
libGLESV2.dll	
opengl32sw.dll	
Qt5Core.dll	
Qt5Gui.dll	
Qt5Network.dll	
Qt5Svg.dll	
Qt5Widgets.dll	
D3Dcompiler_47.dll	
CameraLibrary2013x64S.dll	OptiTrack Camera SDK library
See OpenCV3.2 documentation[4]	OpenCV 3.2 static libraries
visualtest.exe	Program of the Optitrack camera SDK that shows the camera image.
help.pdf	This document that is shown when clicking on *Help* in *Rigid Track*.
markerStandard.xml	Standard marker configuration file. Use this as template for new configurations.
referenceData.xml	Pose of the camera relative to the ground calibration frame, hence navigation coordinate system. This file is the result of 3.5.2.
calibration.xml	Camera matrix and distortion coefficients obtained by a previous calibration, see section 3.5.
pattern.png	Calibration pattern that has to be printed out, see section 3.5.
simpleplot.m	Matlab script for plotting logged data.

Table 3-4: *Rigid Track* source and auxiliary files

```
// main inizialised the GUI and values for the marker position etc
int main(int argc, char *argv[])

// convert a opencv matrix that represents a picture to a Qt Pixmap object
QPixmap Mat2QPixmap(cv::Mat src)

// calculate the chess board corner positions, used for the camera calibration
void calcBoardCornerPositions(Size boardSize, float squareSize, std::vector<Point3f>& corners)

// get the euler angles from a rotation matrix
void getEulerAngles(Mat &rotCamerMatrix, Vec3d &eulerAngles)

// start the loop that fetches frames, computes the position etc and sends it to the object
int start_camera()

// Start or stop the camera depending on if the camera is currently running or not
void start_stopCamera()

// determine the initial position of the object that serves as reference point or as ground frame
    origin
int setZero()

// start the camera calibration routine that computes the camera matrix and distortion coefficients
int calibrate_camera()

// Load a previously saved camera calibration from a file
void load_calibration(int method)

// project some points from 3D to 2D and then check the accuracy of the algorithms
void test_Algorithm()

// project a coordinate CoSy with the rotation and translation of the object for visualization
void projectCoordinateFrame(Mat pictureFrame)

// open the UDP ports for communication
void setUpUDP()

// Add a heading offset to the attitude for the case it is necessary
void setHeadingOffset(double d)

// send the position and attitude over UDP to every receiver, the safety receiver is handled on its own
     in the start_camera function
void sendDataUDP(cv::Vec3d &Position, cv::Vec3d &Euler)

// open the documentation of this software in the system interner browser
void show_Help()

// close the UDP ports again to release network interfaces etc.
void closeUDP()

// load a marker configuration from file. This file has to be created by hand, use the standard marker
    configuration file as template
void loadMarkerConfig(int method)

// draw the position, attitude and reprojection error in the picture
void drawPositionText(cv::Mat &Picture, cv::Vec3d & Position, cv::Vec3d & Euler, double error)

// load the rotation matrix from camera CoSy to ground CoSy
// It is determined during ground calibration and once the camera is mounted and fixed stays the same
void loadCameraPosition()

// get the optimal exposure for the camera. For that find the minimum and maximum exposure were the
    right number of markers are detected
int determineExposure()

// compute the order of the marker points in 2D so they are the same as in the 3D array. Hence marker 1
     must be in first place
// for both, list_points2d and list_points3d
void determineOrder()

// Get the pose of the camera w.r.t the ground calibration frame. This frame sets the navigation frame
    for later results.
// The pose is averaged over 200 samples and then saved in the file referenceData.xml. This routine is
    basically the same as setZero.
int calibrateGround()
```

Listing 3.1: main.cpp functions

3.8.2 UML Activity Diagrams

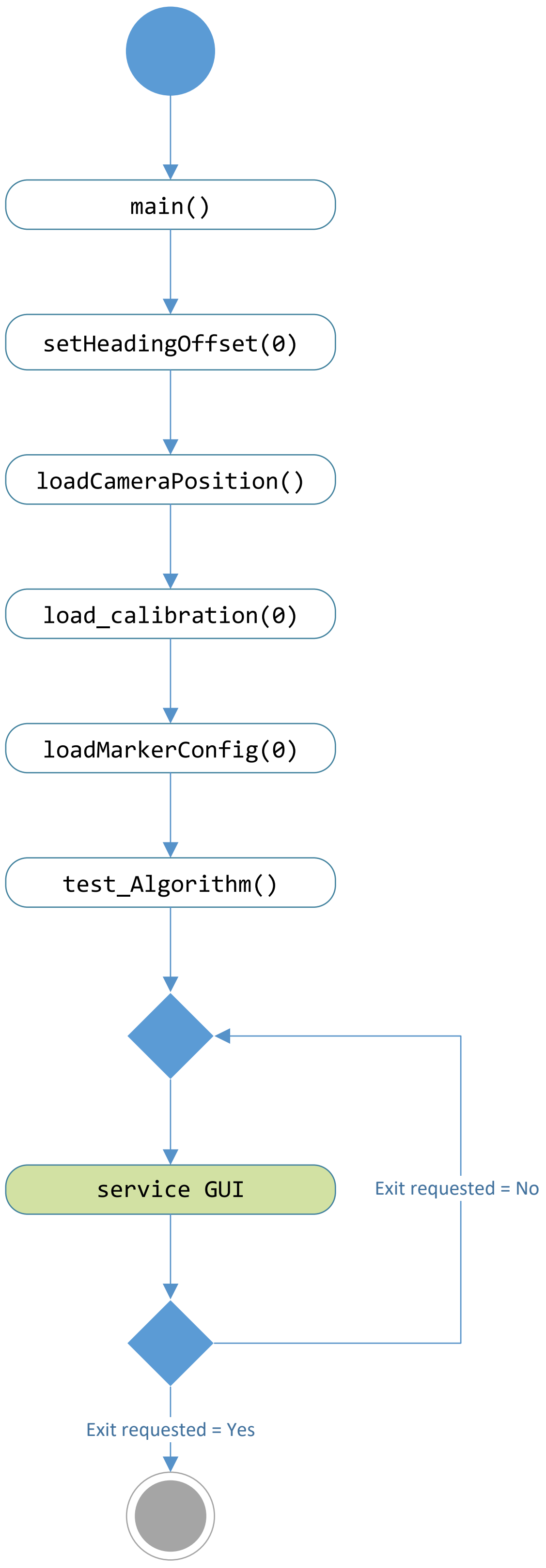


Figure 3-13: *Rigid Track* main function that initializes variables and starts the GUI.

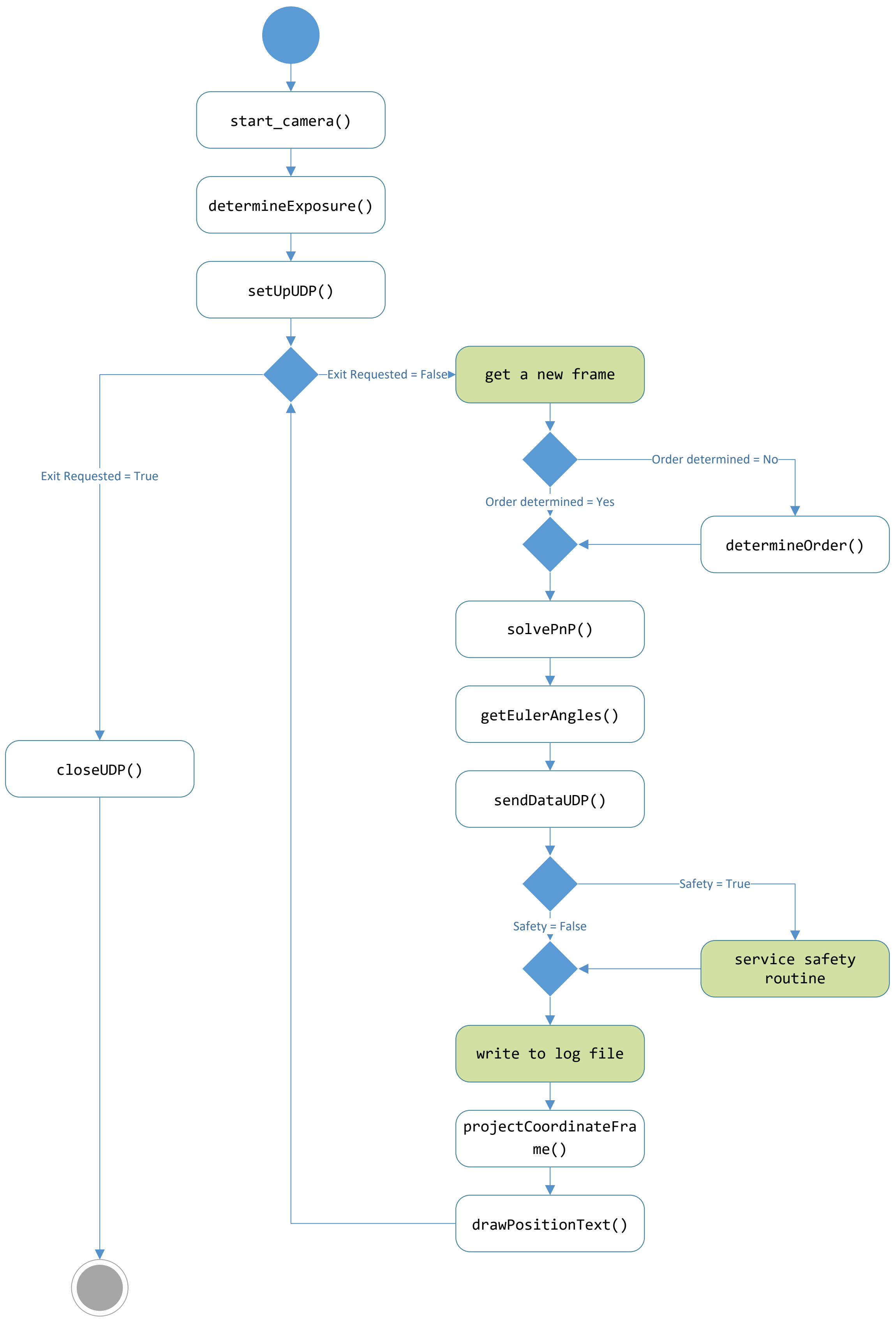


Figure 3-14: *Rigid Track* main function that initializes variables and starts the GUI.

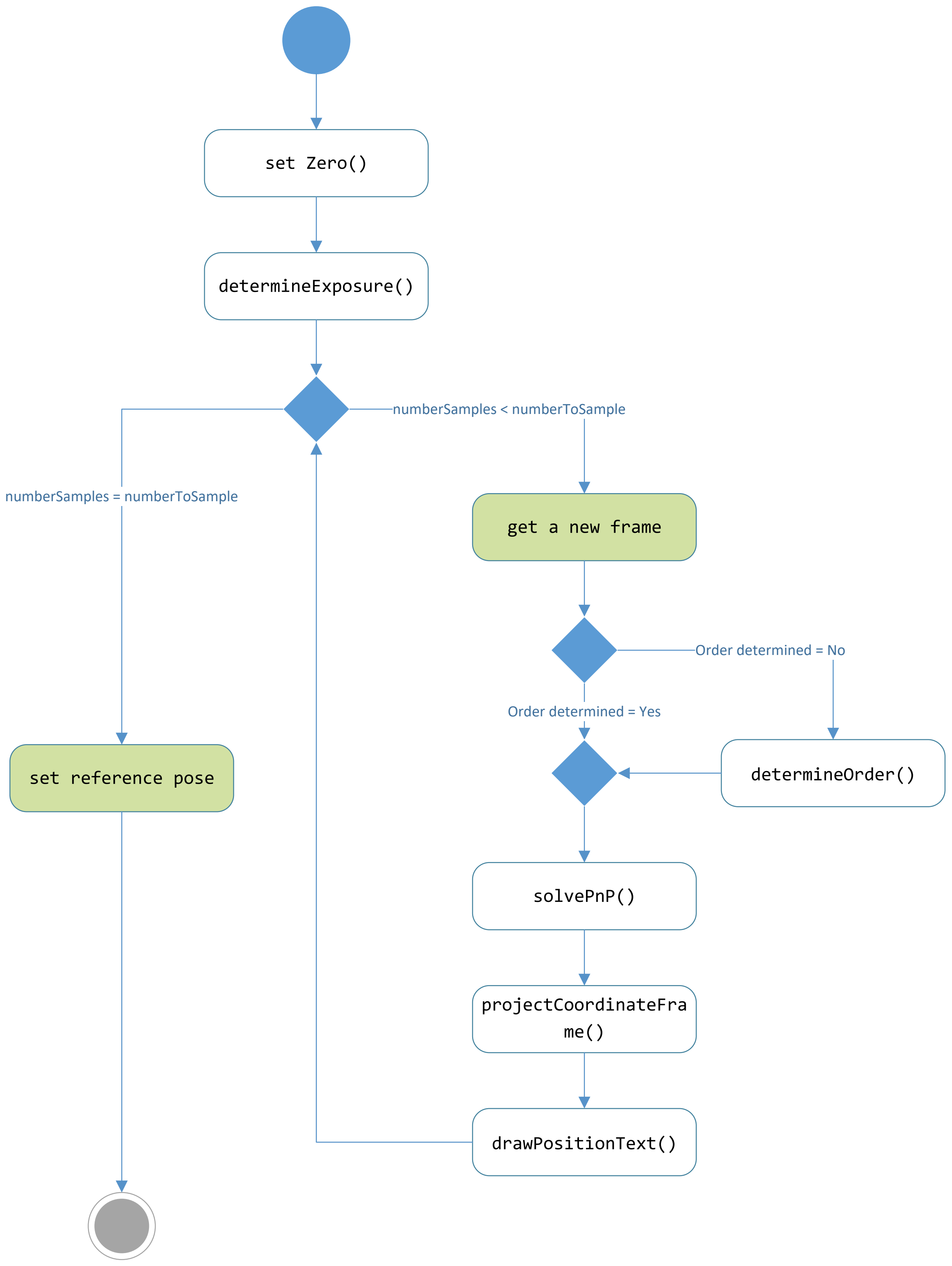


Figure 3-15: *Rigid Track* main function that initializes variables and starts the GUI.

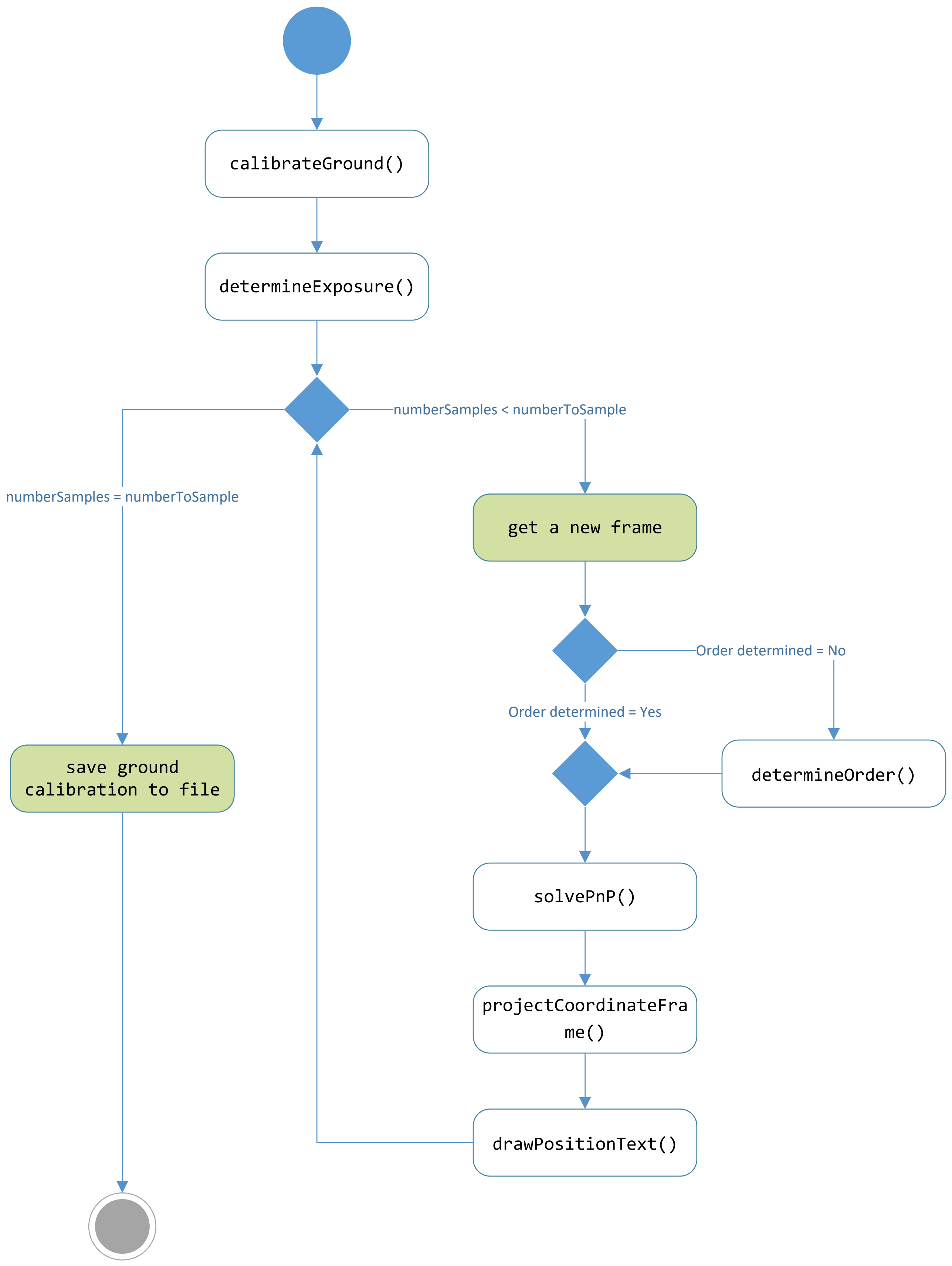


Figure 3-16: ***Rigid Track*** **main function that initializes variables and starts the GUI.**

References

[1] L. Priese, *Computer Vision: Einführung in die Verarbeitung und Analyse digitaler Bilder*. Springer-Verlag, 2015.

[2] (). Optitrack flex 3 in depth, [Online]. Available: `http://optitrack.com/products/flex-3/indepth.html`.

[3] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, “Complete solution classification for the perspective-three-point problem”, *IEEE transactions on pattern analysis and machine intelligence*, vol. 25, no. 8, pp. 930–943, 2003.

[4] V. Lepetit, F. Moreno-Noguer, and P. Fua, “Epnp: An accurate o (n) solution to the pnp problem”, *International journal of computer vision*, vol. 81, no. 2, p. 155, 2009.

[5] G. G. Slabaugh, “Computing euler angles from a rotation matrix”, *Retrieved on August*, vol. 6, no. 2000, pp. 39–63, 1999.

[6] (). Optitrack hardware setup, [Online]. Available: `http://wiki.optitrack.com/index.php?title=Hardware_Setup`.