Universiteit
van
Amsterdam

IRL technical report IRL-UVA-24-01

Automatic Control, Calibration and Recording for the FrodoBots

Qi Zhang and Arnoud Visser
Intelligent Robotics Lab
Universiteit van Amsterdam
The Netherlands

This report introduces the development and implementation of both automatic and manual control systems, camera calibration, and the deployment of state-of-the-art (SOTA) models on Frodobots vehicles equipped with IMU, wheel odometry, cameras, and GPS sensors. We analyze the fundamental parameters of the sensors and employ an Extended Kalman Filter (EKF) to fuse visual odometry (VO) with other odometry data to achieve automated navigation to the greatest extent possible. Additionally, the report discusses the calibration procedures and considerations for actual deployment.

IRL
Intelligent Robotics Lab

Contents

1 Introduction **1**

2 System Overview **1**
2.1 Hardware Components . . . 1
2.2 Computational Resources . . . 1
2.2.1 Limitations and Considerations . . . 2
2.3 Vehicle Specifications . . . 2
2.4 Software Architecture . . . 3

3 Sensors and Data Analysis **3**
3.1 GPS Sensor . . . 3
3.1.1 GPS Errors and Noise Analysis . . . 4
3.2 IMU Data . . . 4
3.2.1 Noise Characteristics . . . 4
3.3 Wheel Odometry . . . 4
3.4 Camera System . . . 5
3.4.1 Camera Calibration . . . 5
3.5 Calibration Algorithm . . . 5

4 Automatic Control System **6**
4.1 Control Algorithm Design . . . 6
4.1.1 Control Algorithm . . . 6
4.2 Extended Kalman Filter (EKF) for Sensor Fusion . . . 7
4.2.1 EKF Algorithm . . . 7
4.3 Visual Odometry (VO) . . . 8
4.3.1 Visual Odometry Algorithm . . . 8

5 Application of SOTA Models on FrodoBots-2K Dataset **9**
5.1 Visual SLAM with ORB-SLAM3 . . . 9
5.1.1 ORB-SLAM3 Algorithm . . . 9
5.2 Object Detection with YOLOX . . . 10
5.2.1 YOLOX Algorithm . . . 10
5.3 Depth Estimation with Lite-Mono . . . 11
5.3.1 Lite-Mono Algorithm . . . 11

6 Real FrodoBots Deployments **12**
6.1 Applying Visual SLAM in Real Deployment . . . 12
6.2 Agile Control of Robots . . . 12

7 Experimental Results **13**
7.1 Calibration Accuracy . . . 13
7.1.1 Calibration Evaluation . . . 13
7.1.2 Data Acquisition and Refinement . . . 13

7.1.3 Image Undistortion Results . . . 15
7.1.4 Ablation study . . . 16
7.1.5 Resumé . . . 16
7.2 Real World Control . . . 16
7.2.1 Automatic Control . . . 17
7.2.2 Manual Control . . . 17
7.3 Recording . . . 19

8 Discussion **21**

9 Conclusion **21**

A Sample Scripts and Commands **24**
A.1 Merge Video Frames Script (`merge_ts_files.sh`) . . . 24
A.2 Calibration Script (`auto_calibration1.py`) . . . 24
A.3 Environment Setup for Trajectory Estimation . . . 24
A.4 Advanced Calibration with ArUco Chessboards . . . 25

Intelligent Robotics Lab
Informatics Institute, Faculty of Science
University of Amsterdam
Science Park 900, 1098XH Amsterdam
The Netherlands

www.intelligentroboticslab.nl

Corresponding author:

Q. Zhang
tel: +31 20 525
`q.zhang2@uva.nl`

1 Introduction

This report provides a detailed overview of the algorithms used to enhance EarthRover's autonomous control, perform precise camera calibration, and design data recording methodologies. It also explores methods for applying advanced visual SLAM, object detection, and depth estimation to the dataset. Our focus is on reducing GPS errors, managing sensor noise, and improving localization accuracy by utilizing an Extended Kalman Filter (EKF) for sensor fusion. The goal is to automate dataset recording as much as possible and to demonstrate some challenges compared to existing methods [4].

2 System Overview

The Frodobot's EarthRover Zero is a lightweight robotic platform designed for autonomous navigation and perception tasks. This section provides an overview of the hardware components, computational resources, and the software architecture utilized in this project.

2.1 Hardware Components

- **Frodobots EarthRover Zero**: A compact robotic platform from Frodobots (`https://shop.frodobots.com/`).
- **Inertial Measurement Unit (IMU)**: Provides 9-degree-of-freedom (DOF) data, including accelerometer, gyroscope, and magnetometer readings.
- **Wheel Encoders**: Measure the rotational speed (RPM) of each wheel, providing wheel odometry data.
- **Cameras**:
  - **Front Camera**: Captures video at 20 FPS with a resolution of 1024x576 pixels.
  - **Rear Camera**: Captures video at 20 FPS with a resolution of 540x360 pixels.
- **Global Positioning System (GPS)**: Provides latitude and longitude data at 1 Hz frequency.
- **Microphone and Speaker**: Used for audio input and output at a sampling rate of 16000 Hz.

2.2 Computational Resources

The algorithms were tested on a single Frodobots car using a laptop with the following specifications:

- **Processor**: AMD Ryzen 9 7945HX

- **Graphics Card**: NVIDIA GeForce RTX 3070
- **Memory**: [Specify the amount of RAM, e.g., 32 GB DDR4]

Due to the limited computational resources of the laptop, real-time processing was challenging. We also considered using more powerful GPUs like the NVIDIA RTX 4090 or H100 available on our cluster computers. However, the communication time between the robot and the cluster posed a significant constraint, potentially introducing delays that are detrimental to real-time control.

2.2.1 Limitations and Considerations

- **Processing Power**: The RTX 3070 provides limited computational capabilities compared to higher-end GPUs, affecting the performance of computationally intensive tasks like visual odometry.
- **Communication Latency**: Offloading computations to a remote cluster introduces latency due to network communication times, which can negatively impact the responsiveness of the control system.
- **Real-Time Requirements**: Autonomous navigation requires timely processing of sensor data and execution of control commands. Balancing computational demands with hardware limitations is essential.

2.3 Vehicle Specifications

- **Weight**: Less than 5 kg (11 lbs).
- **Maximum Speed**: The robot reaches its maximum speed of approximately 3 m/s within the first second when the linear velocity command is set to 1. Afterwards, it maintains a steady speed of 3 m/s (or 2.7 m/s in outdoor conditions) from the second onward. During turns, when the linear and angular velocities are set to 1, the linear speed decreases to 2 m/s.
- **Mobility**: Capable of moving forward/backward and turning in place.
- **Connectivity**: Equipped with 4G connection for consistent data transmission across different environments.
- **Stability**: The camera's stability is somewhat poor, likely due to the high position of the camera and the soft material of the connecting bridge. A metal support was added to the soft bridge of the original design, but some camera shake persists, undetected by the IMU. This issue can be mitigated with improved estimation through visual odometry.

2.4 Software Architecture

The software system is built upon the official SDK provided for remote access and control of the robot. The architecture includes:

- **Control Interface**: Receives control commands and sends them to the robot.
- **Sensor Data Acquisition**: Collects data from the GPS, IMU, wheel encoders, and cameras.
- **State Estimation Module**: Implements the EKF for sensor fusion.
- **Visual Odometry Module**: Processes camera images to estimate motion.
- **Navigation and Control Module**: Determines control actions based on the estimated state and target positions.

A block diagram of the system architecture is shown in Figure 1.

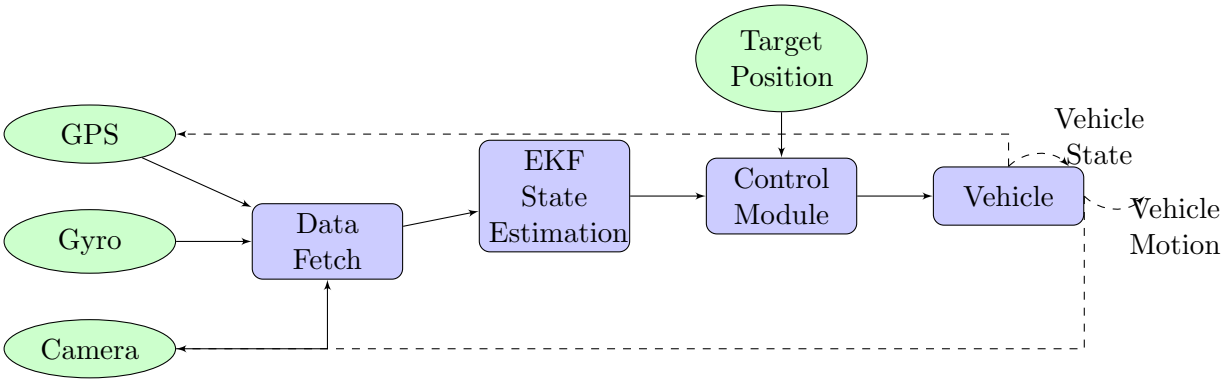


Figure 1: System Summary

3 Sensors and Data Analysis

Understanding the characteristics and limitations of the sensors is crucial for accurate state estimation and control. This section analyzes the data from each sensor and discusses their noise characteristics.

3.1 GPS Sensor

The GPS provides latitude and longitude readings at a frequency of 1 Hz. However, GPS data is susceptible to various errors and noise [1], which can significantly impact localization accuracy.

3.1.1 GPS Errors and Noise Analysis

GPS errors arise from factors such as:

- **Satellite Geometry**: Poor satellite positioning can degrade accuracy.
- **Atmospheric Effects**: Ionospheric and tropospheric delays affect signal propagation.
- **Multipath Effects**: Signals reflecting off surfaces cause inaccuracies.

The public website shows the GPS "The robot is deemed to have successfully reached the next checkpoint if it comes within 15 meters of that point, allowing for the tolerance of noisy GPS data.".

3.2 IMU Data

The IMU provides accelerometer data at 100 Hz, gyroscope data at 1 Hz, and magnetometer data at 1 Hz. The accelerometer and gyroscope data are used for estimating linear and angular velocities.

3.2.1 Noise Characteristics

The IMU data contains high-frequency noise, particularly in the accelerometer readings. A noise model is developed based on the sensor specifications and observed data.

3.3 Wheel Odometry

Wheel encoders measure the RPM of each wheel at an ideal frequency of 10 Hz. Wheel slip and uneven terrain can introduce errors in odometry calculations.

3.4 Camera System

The front and rear cameras capture images used for visual odometry and obstacle detection.

3.4.1 Camera Calibration

Camera calibration is performed with OpenCV, based on Zhang's method [5]. This provides the intrinsic parameters of the camera model, which are essential for accurate visual odometry and depth estimation.

Calibration Procedure:

1. Prepare a checkerboard grid and fix it in front of the robot.
2. Use the robot's camera to capture images of the checkerboard from different angles and positions.
3. Apply calibration algorithms to compute the camera's intrinsic and extrinsic parameters.

3.5 Calibration Algorithm

Algorithm 1 Camera Calibration Procedure

Require: Checkerboard pattern fixed in front of the robot
Ensure: Calibrated camera parameters
Initialize calibration process
while Calibration not complete **do**
    Capture image from the camera
    Detect checkerboard corners
    if Corners detected **then**
        Add image points and object points for calibration
    end if
    Move robot to a new position
end while
Compute camera matrix K and distortion coefficients D using collected data
return K, D

Calibration Results:

The calibrated parameters include the camera matrix K and distortion coefficients D:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad D = [k_1, k_2, p_1, p_2, k_3]$$

These parameters are crucial for undistorting images and ensuring accurate measurements in visual odometry and depth estimation.

4 Automatic Control System

The control system is responsible for navigating the vehicle towards predefined checkpoints while avoiding obstacles and compensating for sensor inaccuracies.

4.1 Control Algorithm Design

The control algorithm employs a combination of proportional controllers for linear and angular velocities.

4.1.1 Control Algorithm

Algorithm 2 Vehicle Control Algorithm

Require: Current state $\mathbf{x}$, Target position $(\phi_{\text{target}}, \lambda_{\text{target}})$
Ensure: Control commands v, ω
1: Compute distance error d between current position and target
2: Compute desired bearing θ_{desired} using:

$$\theta_{\text{desired}} = \arctan 2\left(\sin(\Delta\lambda)\cos(\phi_{\text{target}}), \cos(\phi)\sin(\phi_{\text{target}}) - \sin(\phi)\cos(\phi_{\text{target}})\cos(\Delta\lambda)\right)$$

3: Compute heading error $\Delta\theta = \theta_{\text{desired}} - \theta$
4: Normalize $\Delta\theta$ to $[-\pi, \pi]$
5: **if** $|\Delta\theta| > \theta_{\text{threshold}}$ **then**
6: $\quad v \leftarrow 0$
7: $\quad \omega \leftarrow K_\theta \cdot \Delta\theta$
8: **else**
9: $\quad v \leftarrow K_p \cdot d$
10: $\quad \omega \leftarrow 0$
11: **end if**
12: Limit v and ω to maximum allowed values
13: **return** v, ω

4.2 Extended Kalman Filter (EKF) for Sensor Fusion

The EKF is utilized to fuse data from the GPS, IMU, wheel odometry, and visual odometry to provide an accurate estimate of the vehicle's state [2].

4.2.1 EKF Algorithm

Algorithm 3 Extended Kalman Filter (EKF)

Require: Δt, Initial state $\mathbf{x}_0$, Initial covariance $\mathbf{P}_0$

1: Initialize process noise covariance $\mathbf{Q}$
2: Initialize measurement noise covariance $\mathbf{R}$
3: **for** each time step k **do**
4: **Prediction Step**:

- Receive control input $\mathbf{u}_{k-1} = [v_{k-1}, \omega_{k-1}]^\top$
- Predict state $\mathbf{x}_{k|k-1}$ using motion model
- Compute Jacobian $\mathbf{F}_{k-1}$
- Predict covariance $\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1}\mathbf{F}_{k-1}^\top + \mathbf{Q}$

5: **Update Step**:

- Receive measurement $\mathbf{z}_k$
- Compute innovation $\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}_k\mathbf{x}_{k|k-1}$
- Compute innovation covariance $\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^\top + \mathbf{R}$
- Compute Kalman gain $\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^\top\mathbf{S}_k^{-1}$
- Update state estimate $\mathbf{x}_k = \mathbf{x}_{k|k-1} + \mathbf{K}_k\mathbf{y}_k$
- Update covariance estimate $\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$

6: **end for**

4.3 Visual Odometry (VO)

Visual Odometry estimates the vehicle's motion by analyzing consecutive camera frames [3].

4.3.1 Visual Odometry Algorithm

Algorithm 4 Visual Odometry

Require: Previous image I_{k-1}, Current image I_k, Camera matrix K, Distortion coefficients D

Ensure: Rotation R, Translation t

Undistort I_{k-1} and I_k using K and D
Detect ORB features in I_{k-1} and I_k
Compute descriptors for detected features
Match features between I_{k-1} and I_k using brute-force matcher
Apply ratio test to select good matches
Extract matched keypoints
Compute Essential matrix E using RANSAC
if E is valid **then**
    Recover pose (R, t) from E
    return R, t
else
    return Failure
end if

5 Application of SOTA Models on FrodoBots-2K Dataset

We utilized the FrodoBots-2K dataset to test and evaluate state-of-the-art models for visual SLAM, object detection, and depth estimation. The dataset provides discrete video frames, sensor data, and calibration files.

5.1 Visual SLAM with ORB-SLAM3

5.1.1 ORB-SLAM3 Algorithm

ORB-SLAM3 is a feature-based SLAM system that uses ORB features for tracking and mapping.

Figure 2: ORBSLAM3 applied in the Frodo Bot

Algorithm 5 ORB-SLAM3 Workflow

Require: Video frames, Camera calibration parameters
Ensure: Estimated camera trajectory and map

```
Initialize ORB-SLAM3 system
for each frame I_k do
    Extract ORB features
    Match features with previous frame
    Estimate camera pose
    Update map with new observations
    Perform local mapping and loop closure detection
end for
return Camera trajectory and 3D map
```

5.2 Object Detection with YOLOX

5.2.1 YOLOX Algorithm

YOLOX is a real-time object detection model based on the YOLO (You Only Look Once) family.

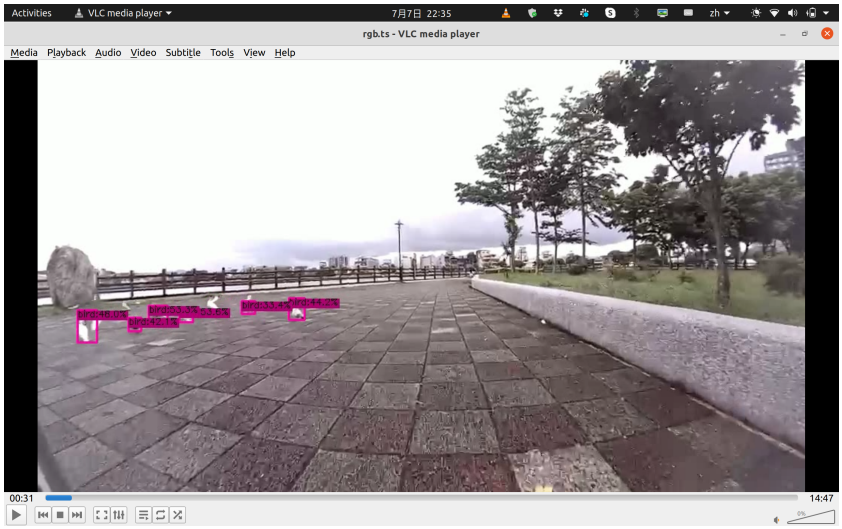


Figure 3: YOLOX applied in the Frodo Bot

Algorithm 6 YOLOX Object Detection

Require: Input image I
Ensure: Detected objects with bounding boxes and class labels
1: Preprocess image I to the required input size
2: Feedforward through YOLOX network
3: Obtain feature maps from the network
4: Apply detection head to predict bounding boxes and class probabilities
5: Perform Non-Maximum Suppression (NMS) to eliminate redundant boxes
6: **return** Detected objects

5.3 Depth Estimation with Lite-Mono

5.3.1 Lite-Mono Algorithm

Lite-Mono is a lightweight monocular depth estimation model designed for real-time applications.

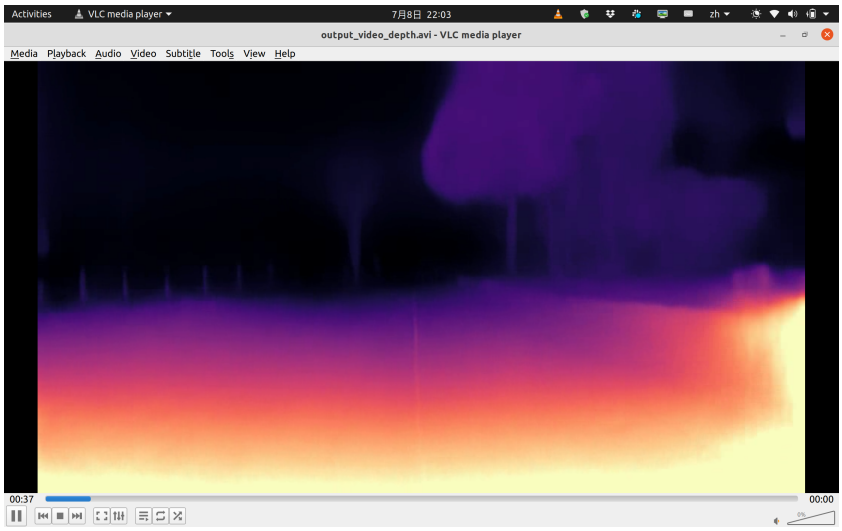


Figure 4: Lite-Mono applied in the Frodo Bot

Algorithm 7 Lite-Mono Depth Estimation

Require: Input image sequence $\{I_t\}$
Ensure: Estimated depth maps $\{D_t\}$
1: **for** each frame I_t **do**
2: Preprocess image I_t
3: Feedforward through Lite-Mono network
4: Obtain depth map D_t
5: **end for**
6: **return** Depth maps $\{D_t\}$

6 Real FrodoBots Deployments

In addition to dataset testing, we deployed the algorithms on real Frodobots hardware.

6.1 Applying Visual SLAM in Real Deployment

Algorithm 8 Deploying ORB-SLAM3 on Frodobots

Require: Live camera feed from robot, Camera calibration parameters
Ensure: Real-time camera trajectory estimation
1: Initialize ORB-SLAM3 system with live camera feed
2: **while** Robot is operational **do**
3: Capture frame I_k from camera
4: Perform visual SLAM processing as per Algorithm 5
5: Use estimated pose for navigation and control
6: **end while**

6.2 Agile Control of Robots

We implemented an agile control mechanism to send commands to the robot at a higher frequency.

Algorithm 9 Agile Robot Control

Require: Desired control inputs v, ω
Ensure: Smooth and responsive robot motion
1: **while** Robot is operational **do**
2: Send control commands v, ω to robot
3: Wait for control interval (e.g., 0.05s)
4: **end while**

7 Experimental Results

Experiments were conducted to evaluate the performance of the control algorithms, calibration procedures, and the application of SOTA models.

7.1 Calibration Accuracy

The calibration process resulted in accurate intrinsic parameters, significantly enhancing the performance of visual odometry (VO) and depth estimation. To achieve robust calibration, we employed a systematic approach involving the generation of diverse calibration images and meticulous data refinement.

7.1.1 Calibration Evaluation

- **Reprojection Error**: The mean reprojection error achieved was less than one pixel, indicating high calibration accuracy.(Error in pixels)
- **Impact on VO**: Improved calibration reduced drift in visual odometry.

7.1.2 Data Acquisition and Refinement

To ensure comprehensive coverage of various viewing angles during calibration, the vehicle was subjected to random linear and angular velocities. This maneuvering facilitated the capture of calibration images from multiple perspectives, enhancing the robustness of the calibration process.

Post-capture, a manual filtering step was implemented to remove redundant images (images with repeat patterns and texture) and those lacking a complete chessboard pattern. This refinement significantly improved calibration accuracy by ensuring that only high-quality, informative images contributed to the calibration parameters.

The parameters which are calibrated with Zhang's method [5] are the camera intrinsic matrix and the distortion coefficients. The camera intrinsic matrix is assumes the geometric transformation of a pinhole, and estimates 4 parameters (f_x,f_y,c_x,c_y), which are respectively the camera's focal length (f_x,f_y) and the camera's optical center (c_x,c_y) (both in defined in pixels):

$$\begin{bmatrix} f_x & 0.0 & c_x \\ 0.0 & f_y & c_y \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

The images coming from the front fish-eye camera show a clear barrel distortion (see Fig. 6. Zhang's method [5] also estimates the the radial and tangential part of the distortion (k_1,k_2,p_1,p_2,k_3). The radial distortion is estimated up to the 3^{th} order coefficient, with the distorted pixel point (x,y) is displaced as function of the distance $r=\sqrt{x^2+y^2}$ from the camera's optical center:

$$x_{distorted} = x(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

When the image plane is not perfectly parallel to the plane of the lens, tangential distortion occurs. We have found in the calibartion results that the coefficients (p_1, p_2) are small, so minimal tangential correction has to take place.

Calibration Metrics Before Data Refinement Prior to the removal of redundant and incomplete chessboard images, the calibration yielded the following parameters:

- **RMS Reprojection Error**: 1.4140 pixels
- **Camera Matrix**:
$$\begin{bmatrix} 205.846 & 0.000 & 241.400 \\ 0.000 & 206.022 & 155.694 \\ 0.000 & 0.000 & 1.000 \end{bmatrix}$$
- **Distortion Coefficients**:
$$\begin{bmatrix} -0.2164 & 0.0571 & -0.000755 & -0.000892 & -0.007217 \end{bmatrix}$$

Calibration Metrics After Data Refinement After the elimination of redundant and incomplete images, the calibration accuracy improved markedly:

- **Calibration Error**: 0.0773 pixels
- **Camera Matrix**:
$$\begin{bmatrix} 407.860 & 0.000 & 533.301 \\ 0.000 & 407.866 & 278.699 \\ 0.000 & 0.000 & 1.000 \end{bmatrix}$$
- **Distortion Coefficients**:
$$\begin{bmatrix} -0.2172 & 0.0537 & 0.001853 & -0.002105 & -0.006000 \end{bmatrix}$$

These refined parameters demonstrate a significant reduction in calibration error, affirming that the manual filtering process effectively enhances calibration precision. Next to the regular chessboard pattern, we also applied this method on ArUco Chessboards. Unfortunately, as can be seen in Appendix A.4, this method did not improve the calibration accuracy.

7.1.3 Image Undistortion Results

To visually assess the effectiveness of the calibration parameters, we processed images both before and after undistortion using the obtained calibration data. Figures 5 and 6 illustrate the improvements in image quality post-undistortion, highlighting the reduction of lens-induced distortions.

Figure 5: Distorted image before calibration

Figure 6: Undistorted image after calibration

These visual results demonstrate the quantitative improvements in calibration accuracy, demonstrating the practical benefits of precise calibration in reducing image distortions.

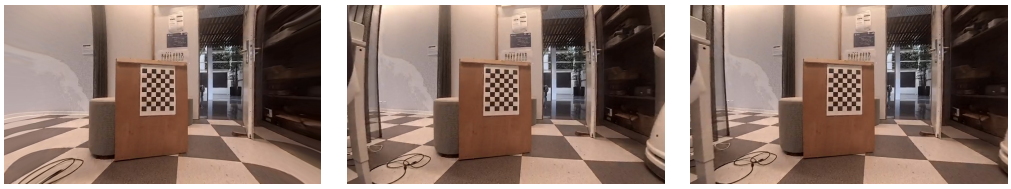
7.1.4 Ablation study

To see the effects of each of the five distortion coefficient (k_1,k_2,p_1,p_2,k_3), we only corrected the lower order components, and left the higher order components zero.

(a) (k_1,k_2) (b) (k_1,k_2,p_1) (c) (k_1,k_2,p_1,p_2)

Figure 7: The effect of tangential coefficients (p_1,p_2) after an 2^{nd} order correction of the radial distortion.

As can be seen in Fig. 8 the effect of tangential coefficients is minimal. The effect of the three radial distortion coefficients is much more distinct effect, for instance when one focuses on the robot at the right or the table at the left:



(a) (k_1) (b) (k_1,k_2) (c) (k_1,k_2,p_1,p_2,k_3)

Figure 8: The effect of radial coefficients (k_1,k_2,k_3) on the distortion correction.

7.1.5 Resumé

Overall, the calibration process, enhanced by strategic data acquisition and refinement, achieved the desired accuracy levels, thereby improving the performance of visual odometry and depth estimation systems. While advanced methods like ArUco-based calibration show promise, further refinements are necessary to match the efficiency and accuracy of traditional chessboard-based approaches.

7.2 Real World Control

In this subsection, we explore the deployment of our robot in real-world environments, focusing on both automatic and manual control strategies. The challenges of navigating using GPS-based targets, especially in varying environments such as indoor and outdoor settings, are addressed. Additionally, we discuss the integration of multiple sensors to enhance navigation accuracy and the practical considerations involved in implementing control algorithms under resource constraints.

7.2.1 Automatic Control

Deploying the robot in real-world scenarios necessitates robust navigation capabilities based on GPS target points. However, indoor environments present significant challenges due to weak GPS signals. To mitigate this, after the initial GPS signal acquisition, we employ multi-sensor fusion techniques to simulate GPS variations. This approach eliminates the need to position the vehicle outdoors solely for GPS initialization.

Given that the robot utilizes a monocular camera setup, scale estimation becomes critical. To achieve accurate scale estimation, we integrate data from an Inertial Measurement Unit (IMU) with the visual inputs. This fusion compensates for the limitations of monocular vision and provides a more reliable estimation of the robot's position and orientation.

Outdoors, GPS signals are inherently subject to inaccuracies, which complicates the tuning of noise parameters in the Kalman filter used for state estimation. Initially, we attempted to set checkpoints based on the robot's GPS-derived latitude and longitude. However, this approach resulted in consistent misalignment of the robot's heading direction. To resolve this, we transitioned to manually selecting corresponding GPS coordinates from Google Maps to define accurate checkpoints.

Furthermore, the robot's orientation is calculated as a rotation angle relative to the North Pole. This orientation estimation conflicts with the path planning framework, which is based on azimuth angles. To ensure consistency, we transformed the orientation data into a unified coordinate system compatible with our path planning algorithms.

Resource limitations posed additional challenges, particularly with the SDK's browser service consuming approximately 30% of the CPU resources. This constraint restricted the implementation of advanced algorithms on a standard laptop, often leading to suboptimal estimation performance. For instance, attempts to publish images to a ROS topic and process them with ORBSLAM3 resulted in the blocking of threads responsible for servicing the robot. While leveraging computational clusters could potentially alleviate processing delays, the introduced latency hindered the robot's ability to make real-time decisions effectively. Given the time constraints before deployment, we opted for a purely manual control approach for dataset recording.

7.2.2 Manual Control

Manual control was adopted primarily due to the aforementioned limitations in automatic control implementations. One significant issue with manual control is the inherent latency in command execution, which can lead to delayed responses and persistent movement in the direction of previous commands. To address this, we implemented a strategy to publish control commands at fixed intervals of every 0.5 seconds. In the absence of new control commands, the system automatically sends stop commands with zero linear and angular velocities. This mechanism ensures that the robot does not continue to accelerate unintentionally and can maintain better control over its movements.

Despite the challenges, manual control provided a reliable method for recording datasets under the given constraints. It allowed us to gather necessary data without

the complexities introduced by real-time sensor fusion and advanced algorithmic processing, facilitating a smoother deployment process within the limited timeframe.

Algorithm 10 Manual Car Control System

Require: Keyboard inputs, HTTP requests
Initialize control URL and log file
function GET_CURRENT_DATA
    Fetch current data via GET request
end function
function LOG_COMMAND(linear, angular)
    Log timestamp, linear, and angular values
end function
function SEND_COMMAND(linear, angular)
    Send control command via POST request
    Log command if successful
end function
Initialize states: `linear_state`, `angular_state`, `emergency_stop`
function COMMAND_SENDER
    while no emergency stop **do**
        Continuously send commands
    end while
end function
function ON_PRESS(key)
    if key is valid **then**
        Adjust states and send command
    end if
end function
function ON_RELEASE(key)
    Reset states and send stop command
end function
Start keyboard listener for controls

7.3 Recording

Effective dataset recording requires careful consideration of the varying requirements of different sensors. Each sensor operates optimally at specific frequencies for capturing timestamps and recording precise data. To achieve this, the following strategies are employed:

- **Sensor-Specific Frequencies:** Different sensors necessitate distinct recording frequencies to ensure accurate data capture. For instance, while some sensors may require high-frequency data acquisition to capture rapid changes, others may function adequately at lower frequencies.

- **IMU Recording Strategy:** To attain finer granularity in the dataset, the Inertial Measurement Unit (IMU) data is recorded at a lower overall frequency. However, acceleration data from the IMU is recorded at a higher frequency to capture more detailed motion dynamics. This dual-frequency approach balances the need for detailed acceleration information with the efficiency of lower-frequency recordings for other IMU data.

- **Dataset Format:** Adopting a dataset format similar to the TUM (Technical University of Munich) dataset provides a more universal and widely accepted structure[1]. The TUM format is recognized for its flexibility and compatibility with various data processing and analysis tools, making it a suitable choice for diverse applications.

- **Asynchronous Design Implementation:** Notably, SDK version 4.4 lacks support for multithreading operations, which are traditionally used to handle concurrent data recording tasks. To circumvent this limitation, an asynchronous coroutine design is implemented. This approach allows for non-blocking, concurrent execution of recording tasks, effectively serving as the best alternative to multithreading in this context.

By tailoring the recording frequencies to the specific needs of each sensor and adopting a robust dataset format, the recording process ensures high-quality and versatile data collection. The asynchronous coroutine design further enhances the system's ability to manage concurrent tasks efficiently, compensating for the limitations of the existing SDK.

[1] Frodo Dataset Conversion Scripts. Available at `https://github.com/catglossop/frodo_dataset`

Algorithm 11 Asynchronous Data Collection System

Configuration:
  BASE_URL ← `http://localhost:8000`
  DATA_DIR ← `TUM_Dataset`
  INTERVALS ← { `control`: 0.1, `gps`: 1.0, `imu_accel`: 0.01, `imu_gyro`: 1.0, `imu_mag`: 1.0, `camera_rear`: 0.05, `camera_front`: 0.05, `rpm`: 0.1 }
Initialization:
  Create directories under `DATA_DIR`
  Initialize CSV files with headers
Functions:
function SaveControl(session)
  while True **do**
    Fetch control data from `BASE_URL/data`
    if Valid **then**
      Append to `control.csv`
      Log success
    end if
    Sleep `INTERVALS['control']`
  end while
end function
function SaveGPS(session)
  while True **do**
    Fetch GPS data from `BASE_URL/data`
    if Valid **then**
      Append to `gps.csv`
      Log success
    end if
    Sleep `INTERVALS['gps']`
  end while
end function
function Main
  Initialize environment
  Start SaveControl(session)
  Start SaveGPS(session)
  ...
  Await tasks
end function
Execution:
if Main Program **then**
  Main
end if

8 Discussion

The next steps should involve using approximate visual observation as the ground truth for the orientation of the vehicle at the destination, and then adjusting the noise parameters of the EKF to output rotation commands. This will allow us to evaluate the accuracy of the decisions made by the EKF. Additionally, we can measure the azimuth between the destination and the starting point calculated on the map, and compare it with the azimuth based on decisions made using the vehicle's GPS data, to assess the GPS noise. Through these methods, reinforcement learning can be effectively used to tune parameters and further analyze the vehicle's noise. In this way, a more lightweight and robust algorithm can be designed."

9 Conclusion

This technical report has detailed the comprehensive development and implementation of both automatic and manual control systems, camera calibration processes, and the integration of state-of-the-art (SOTA) models on the Frodobots EarthRover Zero platform. Through meticulous sensor analysis and the application of an Extended Kalman Filter (EKF) for sensor fusion, we achieved significant improvements in the vehicle's localization accuracy and navigation capabilities.

Key accomplishments of this project include:

- **Control Systems Development**: The design and implementation of both automatic and manual control algorithms enabled effective navigation towards predefined checkpoints while mitigating the impact of sensor inaccuracies and environmental challenges.
- **Camera Calibration**: A robust calibration procedure was established, utilizing traditional chessboard patterns and exploring advanced ArUco-based methods. The calibration refinements significantly reduced reprojection errors, enhancing the performance of visual odometry and depth estimation.
- **Sensor Fusion with EKF**: By integrating data from GPS, IMU, wheel odometry, and visual odometry, the EKF provided a reliable state estimation framework, improving the vehicle's ability to navigate accurately despite inherent sensor noise and external disturbances.
- **Application of SOTA Models**: The deployment of ORB-SLAM3, YOLOX, and Lite-Mono on the FrodoBots-2K dataset demonstrated the feasibility and effectiveness of leveraging advanced algorithms for real-time perception and decision-making tasks.
- **Real-World Deployments**: Practical deployments highlighted the challenges of operating under resource constraints and varying environmental conditions. The

transition to manual control for dataset recording underscored the importance of adaptable control strategies in real-world scenarios.

- **Dataset Recording Strategies**: Implementing sensor-specific recording frequencies and an asynchronous coroutine design facilitated high-quality data collection, essential for subsequent analysis and algorithmic improvements.

Despite these successes, the project encountered several challenges, particularly related to computational limitations and the complexity of advanced calibration techniques. The latency introduced by offloading computations to remote clusters impeded real-time control, necessitating a reliance on manual control methods for dataset recording within the project timeline.

Future work will focus on refining the sensor fusion algorithms by incorporating visual observations as ground truth for vehicle orientation and adjusting EKF noise parameters to enhance decision-making accuracy. Additionally, leveraging reinforcement learning to fine-tune these parameters holds promise for developing more lightweight and robust control algorithms. Further exploration into optimizing ArUco-based calibration and reducing computational overhead will also be pursued to fully realize the potential of autonomous navigation on the FrodoBots platform.

In conclusion, this project successfully advanced the autonomous capabilities of the Frodobots EarthRover Zero through integrated control systems, precise sensor calibration, and the application of cutting-edge perceptual models. The insights gained lay a solid foundation for ongoing enhancements and the pursuit of more sophisticated autonomous functionalities in future iterations.

References

[1] E. D. Kaplan and C. Hegarty, *Understanding GPS/GNSS: principles and applications*, Artech house, 2017.

[2] L. McGee, S. Schmidt and G. Smith, "Applications of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle", NASA Technical Report R-135, Tech. Rep, 1962.

[3] D. Nister, O. Naroditsky and J. Bergen, "Visual odometry", in "Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.", volume 1, pp. 652–659, 2004, doi:10.1109/CVPR.2004.1315094.

[4] Q. Zhang, Z. Lin and A. Visser, "An Earth Rover dataset recorded at the ICRA@40 party", preprint arXiv 2407.05735, Jul. 2024.

[5] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations", in "Proc. of the 7th IEEE International Conference on Computer Vision", volume 1, pp. 666–673, 1999.

A Sample Scripts and Commands

This appendix provides sample scripts and commands used throughout the project.

A.1 Merge Video Frames Script (`merge_ts_files.sh`)

```
#!/bin/bash
# Merge .ts video files into a single video
cat recordings/*.ts > merged_video.ts
```

A.2 Calibration Script (`auto_calibration1.py`)

Algorithm 12 Calibration Script Pseudocode

Require: Access to robot's control and camera interfaces
Ensure: Collection of calibration images
1: Initialize robot control and camera modules
2: **for** each desired calibration position **do**
3: Move robot to a new random position
4: Capture image from camera
5: **if** Checkerboard detected in image **then**
6: Save image for calibration
7: **end if**
8: Wait for a short duration
9: **end for**
10: Perform camera calibration using collected images

A.3 Environment Setup for Trajectory Estimation

```
# Create and activate conda environment
conda env create --file traj_est_env.yaml
conda activate traj_est
```

A.4 Advanced Calibration with ArUco Chessboards

In an effort to explore more sophisticated calibration techniques, we experimented with ArUco-based chessboards. ArUco markers offer enhanced detection capabilities, potentially improving calibration accuracy. However, this method presented challenges:

- **Implementation Complexity**: Given that comprehensive official documentation is available only in C++, we implemented the ArUco calibration process using C++.
- **Execution Time**: The ArUco-based method exhibited longer execution times compared to traditional chessboard calibration.
- **Influencing Factors**: Various factors, such as marker detection reliability under different lighting conditions and angles, affected the calibration outcomes.

Calibration Metrics with ArUco Chessboards Similar to the traditional method, we performed calibration both before and after removing redundant and incomplete images.

Before Data Refinement:

- **Reprojection Error**: 67.5747 pixels
- **Camera Matrix**:
$$\begin{bmatrix} 327.803 & 0.000 & 511.500 \\ 0.000 & 129.521 & 287.500 \\ 0.000 & 0.000 & 1.000 \end{bmatrix}$$
- **Distortion Coefficients**:
$$\begin{bmatrix} -0.02556 & 6.5663 \times 10^{-5} & 0.01103 & -0.000469 & -4.2383 \times 10^{-8} \end{bmatrix}$$

After Data Refinement:

- **Reprojection Error**: 34.0430 pixels
- **Camera Matrix**:
$$\begin{bmatrix} 288.758 & 0.000 & 520.736 \\ 0.000 & 23.7255 & 370.020 \\ 0.000 & 0.000 & 1.000 \end{bmatrix}$$
- **Distortion Coefficients**:
$$\begin{bmatrix} -0.003084 & -5.8480 \times 10^{-5} & 0.020078 & -0.000683 & 1.5305 \times 10^{-7} \end{bmatrix}$$

While the reprojection error decreased substantially after data refinement, it remained relatively high compared to the traditional chessboard method. This suggests that, although ArUco markers offer certain advantages, their current implementation may require further optimization to achieve comparable calibration accuracy.

Citation

To cite this work, please use one of the following references:

```
@misc{zhang2024earthroverdatasetrecorded,
      title={An Earth Rover dataset recorded at the ICRA@40 party},
      author={Qi Zhang and Zhihao Lin and Arnoud Visser},
      year={2024},
      month={Jul.},
      eprint={2407.05735},
      archivePrefix={arXiv},
      primaryClass={cs.RO},
      url={https://arxiv.org/abs/2407.05735},
      howpublished={preprint arXiv 2407.05735}
}

@misc{zhang2024frodobots,
      title={Automatic Control, Calibration and Recording for the FrodoBots},
      author={Qi Zhang and Arnoud Visser},
      year={2024},
      month={Sep.},
      url={https://www.intelligentroboticslab.nl/reports-and-theses/},
      howpublished={technical report IRL-UVA-24-01},
}
```

IRL/IAS reports

This report is in the series of IRL technical reports, which is a continuation of the original IAS technical reports. The IRL series editor is Arnoud Visser (A.Visser@uva.nl) The series editor was Bas Terwijn (B.Terwijn@uva.nl). Within this series the following titles appeared:

A. Visser *A Guide to the RoboCup Virtual Rescue Worlds* Technical Report IRL-UVA-16-01, Informatics Institute, University of Amsterdam, The Netherlands, May 2016.

A. Visser *UvA Rescue Technical Report: A description of the methods and algorithms implemented in the UvA Rescue code release* Technical Report IAS-UVA-12-02, Informatics Institute, University of Amsterdam, The Netherlands, December 2012.

A. Visser *A survey of the architecture of the communication library LCM for the monitoring and control of autonomous mobile robots* Technical Report IAS-UVA-12-01, Informatics Institute, University of Amsterdam, The Netherlands, December 2012.

All technical reports are available for download at the IRL website: https://www.intelligentroboticslab.nl/reports-and-theses/.