Master CORO M1
Master Commande et Robotique
Master in Control and Robotics

JEMARO M1
Japan-Europe Master on Advanced Robotics

Project Report
19/06/2022

SLAM with UAV

Nicolás FRANCINELLI
Ke GUO

Supervisor(s)

Damien SIX
Olivier KERMORGANT

Abstract

The aim of this project is to perform visual simultaneous localization and mapping with an UAV (Unmanned Aerial Vehicle). The drone used for this project is a PX4 Vision equipped with a Structure Core sensor. ROS2 is used, along several packages available: (1) the Cartographer package for mapping and localisation, (2) the ls2n_drone_ros2 packages to control drone motions. The first step of the project was to get an understanding of the packages. Then, to use the sensor SDK combined with Cartographer to perform SLAM. And, finally, using the obtained 3D map to track some trajectories with the UAV. The experiments were performed in the LS2N drone arena.

Keywords: SLAM, UAV, ROS2

Contents

1 Introduction

With the development of artificial intelligence and control algorithms and the improvement of sensor accuracy in recent years, the Unmanned Aerial Vehicles(UAVs) known as drones, now are playing an important role in a variety of fields such as survey [1], agriculture [2], urban beautification [3] and many fields. Now there are also multiple types of UAVs on the market including hexacopter [4], quad-rotor [5], fixed-wing [6], and helicopter[7]. Among them, quadrotor UAVs have received a lot of attention from researchers due to their simple structure and potential for small-range hovering, takeoff and landing. Many applications of quadrotor UAVs have been studied for more than 20 years. In particular, vision applications have been demonstrated excitingly, such as rescue [8], target tracking [9], and autonomous landing on a moving unmanned ground vehicle (UGV) [10]. In some of these applications, the vehicle is often required to operate over unknown terrain or where navigation grade terrain maps are unavailable. And the global positioning system(GPS) is unavailable as a localization aid for an airborne vehicle. The use of the SLAM algorithm to detect the environment and to localize the drone in such cases is necessary and a prerequisite to carry out other applications.

SLAM or simultaneous localization and mapping is a challenging problem in mobile robotics that has been taken widely under study for more than two decades where scientists use different techniques to improve autonomy and self-exploration of robot navigation. Over the last few years, many approaches to this problem have been developed, with bundle adjustment [11] or filtering-based estimation [12]. These methods provide an approximate solution but are not exact in the overall sense. In the past two decades, SLAM solving techniques have had a dramatic progression. Various SLAM algorithms are developed with using various sensors such as ultrasonic sensors, laser scanners, RGB cameras and etc for estimating robot's pose and simultaneously building the 2D or 3D maps. It is said that the 2D SLAM problem with using range finders is thought out as a solved problem [13], it makes more scientists working on 3D SLAM. In recent years, some RGB-D cameras like Microsoft Kinect have been interested because of their ability to provide real-time color images and depth maps. Now some approaches for SLAM combine the scale information of 3D depth sensing with the visual information of the cameras to create accurate 3D environment maps and use dense visual odometry [14] [15].

Our approach uses the odometry which is created by EKF(External Kalman Filter) algorithm, 3D depth points cloud which is provided by the Structure Core camera and the camera's IMU (Inertial Measurement Unit) synchronously to create the 3D map. It should be noted that the odometry is generated by the EKF algorithm using the MOCAP (Motion Capture) data and the drone's own IMU.

2 Lab Environment

2.1 Experimental field

The experiment is performed in the LS2N drone arena which is a $6 \times 3.5 \times 4\ m^3$ area with mesh guardrails and 8 MOCAP cameras as shown in Fig.1.

Figure 1: LS2N drone arena

2.2 PX4 Vision

PX4 Vision is a drone kit which contains a near-ready-to-fly carbon-fiber quadcopter equipped with a Pixhawk 4 flight controller, UP Core companion computer, and Structure Core depth camera sensor. All its components are shown in Fig.2.

Figure 2: All components of PX4 Vision UAV

The base_link frame, the depth sensor frame and the frame of the IMU of the sensor are shown in Fig.3. The origin of the IMU frame is the same as the origin of the depth sensor, but in the Fig.3, they are displayed separately so they can be clearly identified.

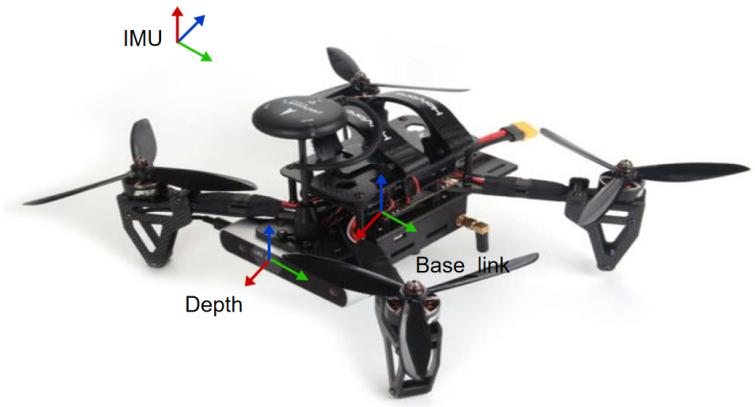


Figure 3: The frames on the PX4 Vision UAV

2.3 ROS2

Robot Operating System (ROS) is an open source robotics middleware suite. Although ROS is not an operating system but a collection of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. ROS1 is built on top of Linux, mainly supporting Ubuntu, while ROS2 adopts a new architecture, based on the underlying DDS(Data Distribution Service) communication mechanism, supporting real-time, embedded, distributed, multi-operating systems. The communication system of ROS1 is based on TCPROS/UDPROS and is strongly dependent on the processing of the master node, while the communication system of ROS2 is based on DDS, which in turn eliminates the master node. ROS2 also has increased programming language version requirements. The most significant change is that the writing of the launch file is very different from ROS1. In this experiment we use the ROS 2 Galactic released in May 2021 on Ubuntu 20.04 as the software development environment [16].

2.4 ROS2 packages

In this experiment we used some packages developed by LS2N lab to control the UAV, to receive the message from the sensor of the drone and to implement the Structure Core camera drive and communication. They are ls2n_drone_ros2 [17] and structure_core_ros2_driver [18].

3 SLAM

3.1 Google Cartographer

Cartographer [19] is a system that provides real-time simultaneous localization and mapping (SLAM) in 2D and 3D across multiple platforms and sensor configurations.

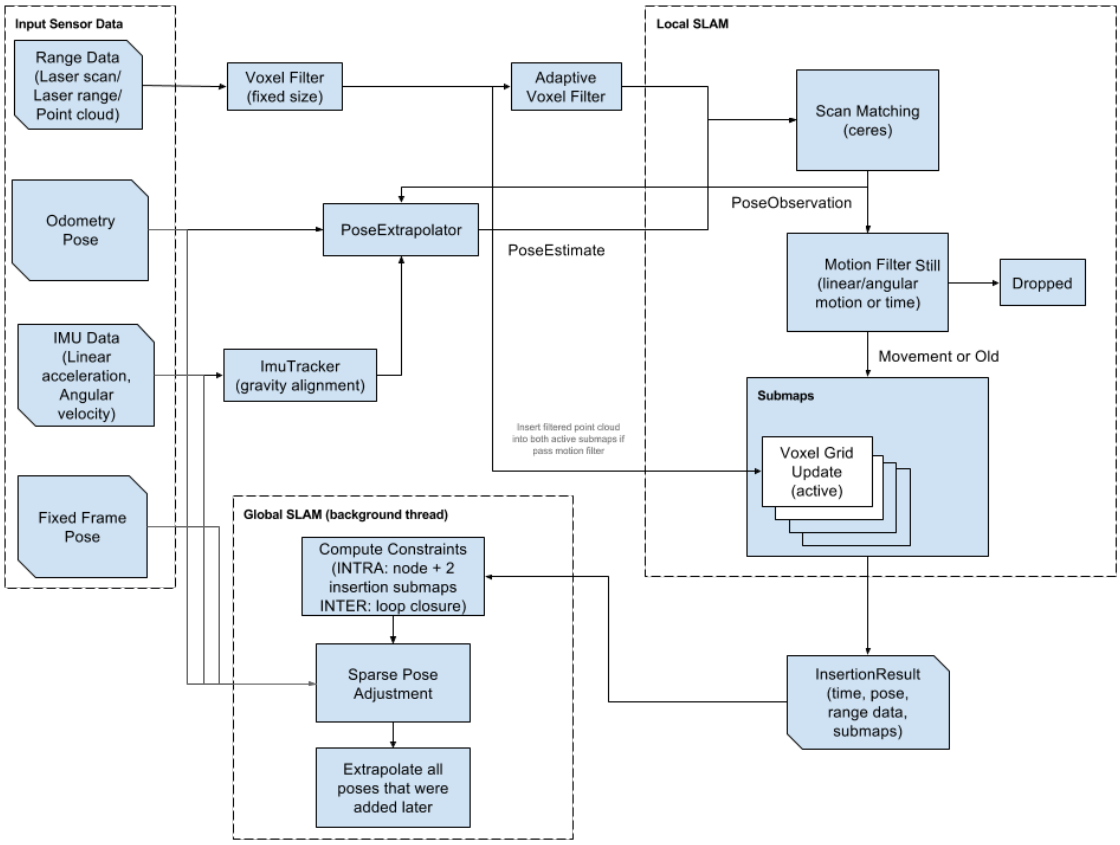


Figure 4: High level system overview of Cartographer

Cartographer is composed of two different, but related subsystems. The first one is local SLAM (sometimes also called front-end or local trajectory builder). Its job is to build a succession of submaps. Each submap is meant to be locally consistent but we accept that local SLAM drifts over time. The other subsystem is global SLAM (sometimes called the back-end). It runs in background threads and its main job is to find loop closure constraints. In other words, its job is to put them all together in the most consistent way.

3.1.1 Inputs

Some of the measurements provided by the range sensors are irrelevant for SLAM. For example, close measurements may come as a result of dust present in the sensor or suspended in the air close to it, likewise, some of the furthest measurements can also come from undesired sources as reflection or sensor noise. To deal with those

issues, Cartographer starts by applying a band-pass filter and only keeps range values between a certain min and max range.

Data from range sensors is typically measured from a single point on the robot but in multiple angles. Close surfaces are hit more often, providing lots of points, compared to objects far away that get hit less times. To reduce computational weight of points handling, a subsample of these point clouds is made. To avoid losing points from far away surfaces, were there is already a low density of measurements, a voxel filter that downsamples raw points into cubes of a constant size and only keeps the centroid of each cube is used. A smaller size cube means better representation but causes more computation. A large cube size will cause data loss, but will be much faster.

After having applied the previously mentioned fixed-size voxel filter, Cartographer also applies an adaptive voxel filter. This filter tries to determine the optimal voxel size (under a max length) to achieve a target number of points. In 3D, two adaptive voxel filters are used to generate a high resolution and a low resolution point clouds.

3.1.2 Local SLAM

After a scan has been filtered, it is passed to the local SLAM algorithm. Local SLAM inserts a new scan into its current submap construction by scan matching using an initial guess from the pose extrapolator, which uses sensors data of other sensors to predict where the next scan should be inserted into the submap.

To perform the scan matching, Cartographer employs CeresScanMatcher. It takes the initial guess as prior and finds the best spot where the scan match fits the submap. It does this by interpolating the submap and sub-pixel aligning the scan. This is fast, but cannot fix errors that are significantly larger than the resolution of the submaps.

A submap is considered as complete when the local SLAM has received a set amount of range data. The issue with local SLAM is that it drifts over time, global SLAM is used to fix this drift. Submaps must be small enough so that the drift inside them is below the resolution, so that they are locally correct. On the other hand, they should be large enough to be distinct for loop closure to work properly.

3.1.3 Global SLAM

While the local SLAM generates the submaps, a global optimization task runs in the background. Its objective is to arrange the set of submaps so that they create a coherent global map. This is done in batches once a certain number of trajectory nodes was inserted.

The global SLAM essentially makes a pose graph optimization which works by building constraints between nodes and submaps and then optimizing the resulting constraints graph. The official documentation offers an intuitive way of thinking

constraints as little ropes tying all nodes together. The resulting net is called the "pose graph".

- Non-global constraints (also known as intra submaps constraints) are built automatically between nodes that are closely following each other on a trajectory. Intuitively, those "non-global ropes" keep the local structure of the trajectory coherent.
- Global constraints (also referred to as loop closure constraints or inter submaps contraints) are regularly searched between a new submap and previous nodes that are considered "close enough" in space (part of a certain search window) and a strong fit (a good match when running scan matching). Intuitively, those "global ropes" introduce knots in the structure and firmly bring two strands closer.

When a node and a submap are considered for constraint building, they go through a first scan matcher called the FastCorrelativeScanMatcher. This scan matcher has been specifically designed for Cartographer and makes real-time loop closures scan matching possible. The FastCorrelativeScanMatcher relies on a "Branch and bound" mechanism to work at different grid resolutions and efficiently eliminate incorrect matchings. This mechanism is extensively presented in [19]. And once the FastCorrelativeScanMatcher has a good enough proposal, it is then fed into a Ceres Scan Matcher to refine the pose.

4 Setting up the environment

To be able to correctly use the cartographer_ros package, first it was needed to publish the static transforms of the frames for the drone and sensors, as well as the transform from the odom frame to base_link frame using the pose obtained by the EKF.

To implement this, first, a ROS2 node (which code can be seen at appendix A) was created with the objective to subscribe to the topic were the EKF publishes the odometry, and after a message is received, the node publishes the transformation. Then, inside a launch file (that can be seen in appendix B) all the static transforms and the odom → base_link transform are published.

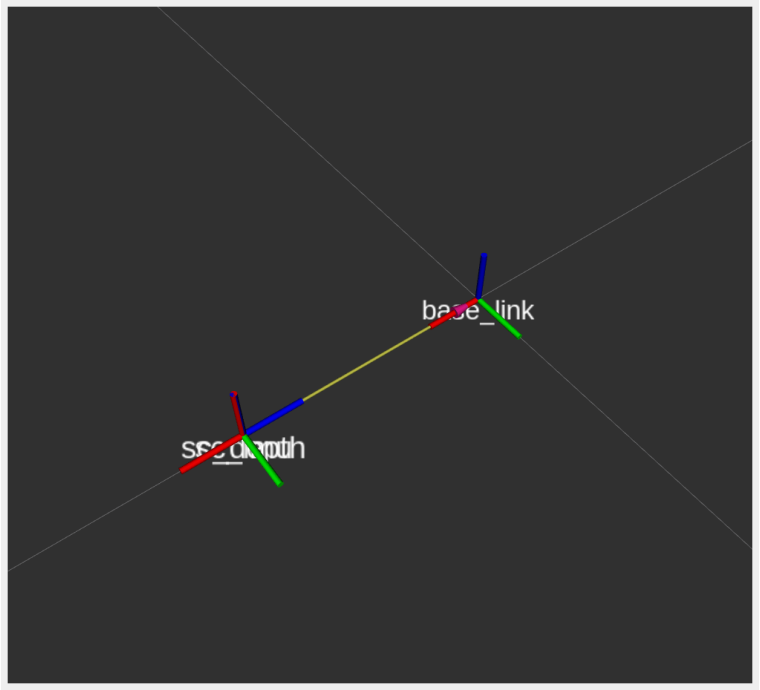
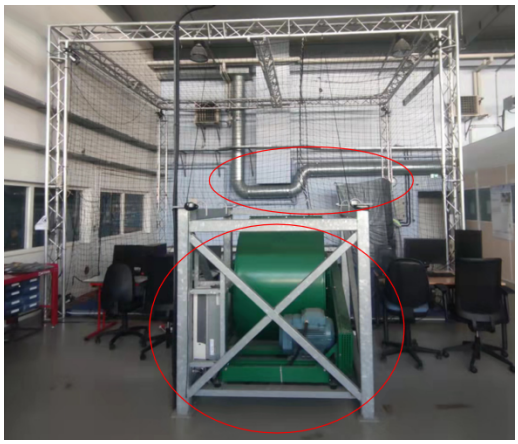


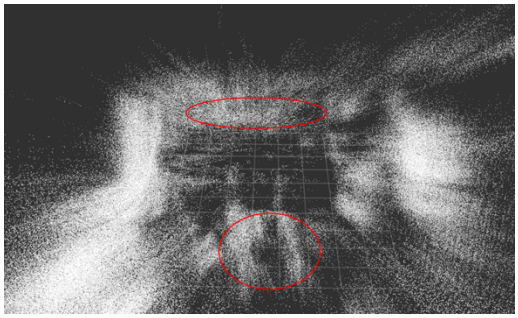
Figure 5: Static transform frames

5 Result and Analysis

Based on the SLAM algorithm introduced in part 3.1, we modified its configuration to make it more adaptable to our experimental requirements and created one launch file to implement all processes. The result is shown in Fig.6 and Fig.7.



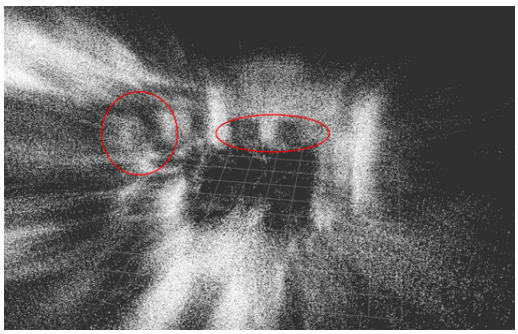
(a) The front view of the lab environment



(b) The front view of the result of the 3D SLAM in rviz

Figure 6: The comparison between the front view of the real lab environment and the result of the 3D SLAM in rviz

(a) The right view of the lab environment



(b) The right view of the result of the 3D SLAM in rviz

Figure 7: The comparison between the right view of the real lab environment and the result of the 3D SLAM in rviz

From the above figures, it can be noticed that the tubes on the wall and that large equipment shown in Fig.6a with red circles are detected. Moreover the windows and the door shown inf Fig.7a with red circles are simulated clearly in the 3D cloud map.

Although the result is acceptable, there is still some room for improvement regarding the performance. Furthermore, because of the lack of the support for the cartographer package to the ROS2 Galactic and the difficulties encountered in handling the drone, this result is currently optimal.

6 Future Work

6.1 Navigation

In the following phase, we could focus on trying different configurations of the depth sensor on the Structure Core camera to get a better quality map and add the covariance to the sensor. After than we can do a whole lab scan to get the 3D cloud map of the whole lab environment. After mapping and localisation, it is the stage of the navigation. Nav2 is the most common package to be used for navigation in ROS, but it is not so good for supporting the control of the drone. We can try to use the Movelt package[20] for navigation of the drone.

6.2 Camera calibration and object tracking

Camera calibration is necessary in 3D computer vision field in order to extract metric information from 2D images. According to the Zhang's method for camera calibration [21] we learned during the COVIS course, we can use photos of the plane as shown in Fig. 8 from different angles to implement the camera calibration to get a better performance.

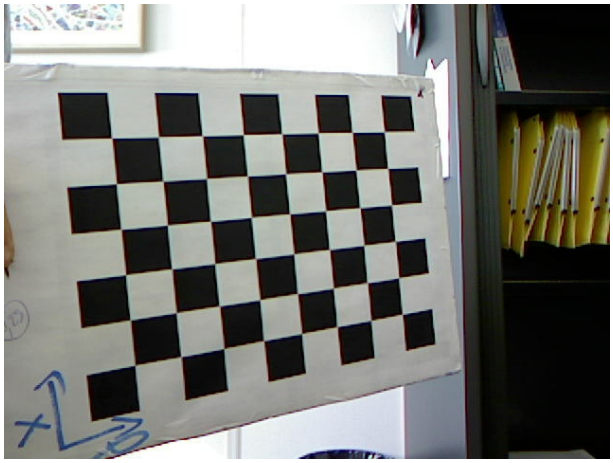


Figure 8: The plane used in Zhang's method for camera calibration

After camera calibration, we can use these validated parameters of the camera for face recognition and object pose determination, converting the pose of the object relative to the camera to a global pose. Going further,this can be combined with the previous navigation algorithm so that many interesting projects could be achieved such as object tracking.

6.3 Offline SLAM

If the environment where the robot is going to work will not change, a better idea is to perform Offline SLAM, which consists on a map learning phase first, and then an operation phase. During the map learning phase, the robot is manually steered through the environment, sensor data is recorded and landmarks are given. Then, in a post-processing step, a map is computed from the recorded data, where the computation does not need to be fast as real-time SLAM, meaning then a better map is possible. In the operation phase it is used for localization, path-planning, etc.

6.4 Improved in a dynamic environment

There are many excellent SLAM systems available, but most of them assume that their working environment is static. When there are dynamic objects in the environment the localization and mapping accuracy of the SLAM system is reduced, and it can even cause its tracking to fail. Therefore there are some discussions on dynamic SLAM. Some scholars propose such an approach which removes feature points of dynamic objects from key frames and then constructs a point cloud map. [22]. In the course of further experiments, this problem and method can be taken into account.

7 Conclusion

In this project, we use PX4 Vision drone to implement a 3D SLAM based on Google Cartographer package, and obtained a 3D cloud point map of the LS2N drone arena.

At the beginning of the project, we worked with the experimental software environment (Ubuntu 20.04 and ROS2 Galactic), and learned about the construction of the drone and its related ros package. Afterwards, we tried to implement a ROS2 package to use the Structure Core depth camera, as the one provided by the manufacturer SDK was not working properly, however, we finished by using one developed by our supervisor. To continue, we performed tests to identify each sensor's frame correctly and created the transformations required to be able to perform SLAM and recorded rosbags to be able to test the SLAM algorithm. Finally, we created the launch and configuration files for Cartographer to obtained the 3D cloud point map of the drone arena.

Even though the resulting point cloud is understandable, the use of ROS2 Galactic imposed by compatibility reasons with the drone, resulted in utilising a cartographer_ros version currently in development, lacking software tools that would have enhanced the quality of the end result. However, the obtained map shows that cartographer is a promising algorithm and should be taken into account for future work once development is finished.

Besides the encountered inconveniences, this project made possible for us to learn about 3D SLAM algorithms and to get a grasp on ROS2, which will be of great help for our professional future.

References

[1] Ian L Turner, Mitchell D. Harley, and Christopher D. Drummond. Uavs for coastal surveying. *Coastal Engineering*, 114:19–24, 2016.

[2] Panagiotis I. Radoglou-Grammatikis, Panagiotis G. Sarigiannidis, Thomas D. Lagkas, and Ioannis D. Moscholios. A compilation of uav applications for precision agriculture. *Comput. Networks*, 172:107148, 2020.

[3] Vishal Verma, Deepali Gupta, Sheifali Gupta, Mudita Uppal, Divya Anand, Arturo Ortega-Mansilla, Fahd S. Alharithi, Jasem Almotiri, and Nitin Goyal. A deep learning-based intelligent garbage detection system using an unmanned aerial vehicle. *Symmetry*, 14:960, 2022.

[4] Valeria Artale, Cristina Lucia Rosa Milazzo, and Angela Ricciardello. Mathematical modeling of hexacopter. *Applied mathematical sciences*, 7:4805–4811, 2013.

[5] Gilar Budi Raharja, Gyu-Beom Kim, and K. J. Yoon. Design of an autonomous hover control system for a small quadrotor. *International Journal of Aeronautical and Space Sciences*, 11:338–344, 2010.

[6] Chao Liang and Chenxiao Cai. Modeling of a rotor/fixed-wing hybrid unmanned aerial vehicle. In *2017 36th Chinese Control Conference (CCC)*, pages 11431–11436, 2017. doi: 10.23919/ChiCC.2017.8029181.

[7] Do-Hyung Kim, Taejoo Kim, Se-Un Jung, and Dong-Il Kwak. Test and simulation of an active vibration control system for helicopter applications. *International Journal of Aeronautical and Space Sciences*, 17:442–453, 2016.

[8] Haider A. F. Almurib, Premeela T. Nathan, and Thulasiraman Nandha Kumar. Control and path planning of quadrotor aerial vehicles for search and rescue. *SICE Annual Conference 2011*, pages 700–705, 2011.

[9] Jose-Ernesto Gomez-Balderas, Gerardo Ramon Flores, Luis Rodolfo García Carrillo, and Rogelio Lozano. Tracking a ground moving target with a quadrotor using switching control. *Journal of Intelligent & Robotic Systems*, 70:65–78, 2013.

[10] Tomá Báa, Petr těpán, Vojtěch Spurný, Daniel Heřt, Robert Pěnika, Martin Saska, Justin R. Thomas, Giuseppe Loianno, and Vijay R. Kumar. Autonomous landing on a moving vehicle with an unmanned aerial vehicle. *Journal of Field Robotics*, 36:874 – 891, 2019.

[11] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Workshop on Vision Algorithms*, 1999.

[12] Stefan Kohlbrecher, Oskar von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. *2011 IEEE*

International Symposium on Safety, Security, and Rescue Robotics, pages 155–160, 2011.

[13] Jiaxin Li, Yingcai Bi, Menglu Lan, Hailong Qin, Mo Shan, Feng Lin, and Ben M. Chen. Real-time simultaneous localization and mapping for uav : A survey. 2016.

[14] Kurt Konolige, Motilal Agrawal, and Joan Solà. Large-scale visual odometry for rough terrain. In *ISRR*, 2007.

[15] Thomas Whelan, Michael Kaess, John J. Leonard, and John B. McDonald. Deformation-based loop closure for large scale dense rgb-d slam. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 548–555, 2013.

[16] https://docs.ros.org/en/foxy/Releases/Release-Galactic-Geochelone.html.

[17] https://gitlab.univ-nantes.fr/ls2n-drones/ls2n_drone_ros2.

[18] https://gitlab.univ-nantes.fr/kermorgant-o/structure_core_ros2_driver.

[19] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016.

[20] https://www.wilselby.com/research/ros-integration/3d-mapping-navigation/.

[21] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1:666–673 vol.1, 1999.

[22] Lanfeng Zhou, Mingyue Kong, Ziwei Liu, and Ling Li. Vision sensor-based slam problem for small uavs in dynamic indoor environments. *Computer Animation and Virtual Worlds*, 2022.

A odom/base_link tf publisher

```cpp
#include <geometry_msgs/msg/transform_stamped.hpp>
#include <nav_msgs/msg/odometry.hpp>

#include <rclcpp/rclcpp.hpp>
#include <tf2/LinearMath/Quaternion.h>
#include <tf2_ros/transform_broadcaster.h>
#include <turtlesim/msg/pose.hpp>

#include <memory>
#include <string>

using std::placeholders::_1;

class FramePublisher : public rclcpp::Node
{
public:
  FramePublisher()
  : Node("tf2_broadcaster_node")
  {
    // Declare and acquire `topic_name_` parameter
    this->declare_parameter<std::string>("topic_name_", "/Drone9/EKF/odom");
    this->get_parameter("topic_name_", topic_name_);

    // Initialize the transform broadcaster
    tf_broadcaster_ =
      std::make_unique<tf2_ros::TransformBroadcaster>(*this);

    subscription_ = this->create_subscription<nav_msgs::msg::Odometry>(
      topic_name_,rclcpp::SensorDataQoS(),
      std::bind(&FramePublisher::handle_drone_pose, this, _1));
  }

private:
  void handle_drone_pose(const nav_msgs::msg::Odometry & msg)
  {
    rclcpp::Time now = this->get_clock()->now();
    geometry_msgs::msg::TransformStamped t;

    // Read message content and assign it to
    // corresponding tf variables
    t.header.stamp = now;
    t.header.frame_id = "odom";
    t.child_frame_id = "base_link";
```

```
    t.transform.translation.x = msg.pose.pose.position.x;
    t.transform.translation.y = msg.pose.pose.position.y;
    t.transform.translation.z = msg.pose.pose.position.z;

    t.transform.rotation.x = msg.pose.pose.orientation.x;
    t.transform.rotation.y = msg.pose.pose.orientation.y;
    t.transform.rotation.z = msg.pose.pose.orientation.z;
    t.transform.rotation.w = msg.pose.pose.orientation.w;

    // Send the transformation
    tf_broadcaster_->sendTransform(t);
  }
  rclcpp::Subscription<nav_msgs::msg::Odometry>::SharedPtr subscription_;
  std::unique_ptr<tf2_ros::TransformBroadcaster> tf_broadcaster_;
  std::string topic_name_;
};

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<FramePublisher>());
  rclcpp::shutdown();
  return 0;
}
```

B tf launch file

```
import os

from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration,TextSubstitution

def generate_launch_description():

    ns = LaunchConfiguration('ns')
    config = LaunchConfiguration('config')

    sc_config_path = os.path.join(get_package_share_directory( 'tf2_broadcaster' ),

    sc_config_file = [TextSubstitution(text=sc_config_path+'/'),
                config, TextSubstitution(text='_config.yaml')]

    os.environ['RCUTILS_CONSOLE_OUTPUT_FORMAT'] = '{time}: [{name}] [{severity}] -
```

```
    imu_frame = [ns, TextSubstitution(text='_imu')]
    rgb_frame = [ns, TextSubstitution(text='_rgb')]
    depth_frame = [ns, TextSubstitution(text='_depth')]
    odom_frame = 'odom'
    drone_frame = 'base_link'

    return LaunchDescription( [
        DeclareLaunchArgument('ns', default_value='sc'),
        DeclareLaunchArgument('config', default_value='sc'),

        #Publish odom -> base_link transform
        Node(package='tf2_broadcaster',
             executable='tf2_broadcaster',
             name='tf2_broadcaster_node',
             output='screen',
             parameters=[
                    {'topic_name_': '/Drone9/EKF/odom'},
                    {'use_sim_time': True}
             ]
        ),
        Node(package='tf2_ros',
             executable='static_transform_publisher',
             name='sc_cam_tf',
             arguments=['0', '0', '0',  '0', '0', '0', depth_frame, rgb_frame],
             parameters=[{'use_sim_time': True}]
        ),

        Node(package='tf2_ros',
             executable='static_transform_publisher',
             name='sc_imu_tf',
             arguments=['0', '0', '0',  '0', '-1.57', '0', depth_frame, imu_frame],
             parameters=[{'use_sim_time': True}]
            ),

        Node(package='tf2_ros',
                  executable='static_transform_publisher',
                  name='sc_drone_cam_tf',
                  arguments=['0.088', '0', '0',  '0', '0', '0', drone_frame, depth_
             parameters=[{'use_sim_time': True}]
                  )
])
```