Integration of Stereo Vision Algorithms in MiRo

Project Report - Software Architecture for Robotics

Parag Khanna and John Thomas

Abstract

The aim of the project was to integrate within the ROS-based MiRo architecture a standard stereo-vision algorithm from the literature and available open-source. The algorithm shall enable MiRo to obtain a 3D representation of the surrounding environment. The report provides a brief description about the MiRo and its ROS-based architecture which was exploited for the implementation of Stereo Vision. The report focuses on the Software Architecture developed for the project and also examines various challenges faced in the implementation at each step. It concludes with future prospects for the Stereo Vision integrated ROS-based architecture with MiRo.

Introduction

Miro is a fully programmable small pet-like autonomous robot for researchers, educators, developers and healthcare professionals. With six senses, eight degrees of freedom, an innovative brain-inspired operating system and a simulation software package, MiRo is a flexible platform suited for developing companion robots, especially for senior citizens or children [1]. The robot is also advertised to be useful in the domains of Human-Robot Interaction (HRI), Robot-Assisted Therapy (RAT), Biomimetics and Brain-Based Robotics, Public Engagement, School and University Teaching [2]. The robot has 2 wheels driven by motors with encoders providing Odometry information. It also has two small cameras on its eyes, each one providing a high-denition image, enabling Stereo-Vision. Apart from MiRo's default control architecture (3BCS, the brain-based biomimetic control system), MiRo can be controlled remotely through WiFi or Bluetooth and can be easily configured as a ROS node. For this, we used the Platform Interface provided. The Platform interface provides direct access to the physical robot, its various sensors and actuators, and its supplementary output systems (sound and light emitters).It is described in Fig. 1.

The schematic of the interface shows the Upstream (data/signals delivered) and Downstream (data/signals received). Before using this, we needed to prepare workstation for MiRo, via installing the MIRO Developer Kit (MDK) and configuring the installation (ROS and/or Gazebo) for use with MiRo [4]. After setting up the Workstation and commissioning the MiRo [5] for use, once roscore is run in the ROS Environment we get a RosNode running which corresponds to various interfaces of MiRo : Standard(new), Platform, Core and Bridge. The Upstream data of platform interface is published as messages over corresponding topics by this node, which can be further subscribed for getting the information from Miro robot. The control signals for the robot can be published as messages from the ROS framework over corresponding topics of Downstream of Platform Interface. Hence we got all the data needed for accessing the information and controlling the MiRo robot.

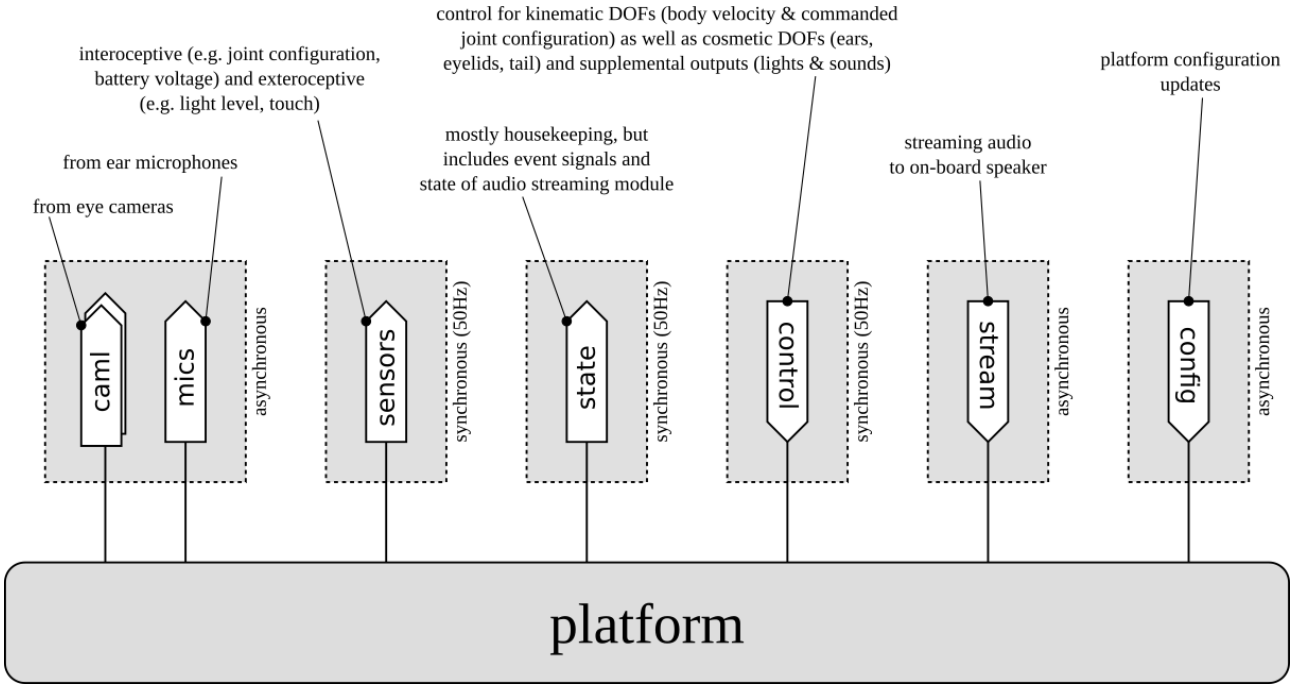

Figure 1: Miro - Platform interface schematic(Reproduced from [3])

Approach

The Miro-mdk publishes various topics including the images from left and right camera over topics:

- miro/rob01/platform/caml
- miro/rob01/platform/camr

After an extensive search for available ROS packages that implements stereo image processing, we decided to use the standard *image_pipeline* stack. The major components of the stack includes [6] :

- Calibration
- Stereo Processing
- Visualization

Camera Calibration

MiRo-mdk did not publish the much required meta information of the cameras in terms of CameraInfo message. Hence the first step was to do calibration of the stereo images. The *camera_calibration* [7]

package was used to find the Camera parameters (saved as .yaml file). A major hurdle in successfully stereo-calibration was the narrow stereo overlap region of 30^0 [10]. OpenCV (CV_bridge for ROS) tools were used to resize the image topics published from MiRo before republishing it. Python scripts [8] were then run to parse data from .yaml file and to publish it as a standard CameraInfo messages. The following pair of images shows an instance of stereo calibration done using the package.

Figure 2: Left image used for calibration.

Figure 3: Right image used for calibration.

Stereo Processing

Stereo processing of the raw images to produce disparity images and point clouds was implemented using the *stereo_image_proc* node [9] of *image_pipeline* stack. Fig. 4 gives a general schematic of the package.

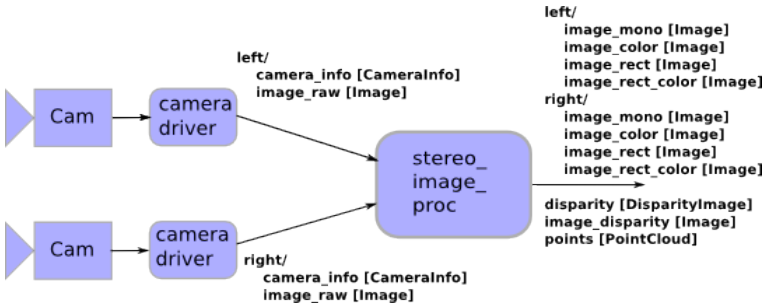

Figure 4: stereo_image_proc - package schematic(Reproduced from [9])

Stereo adapter for MiRo

As shown in Fig. 4, *stereo_image_proc* package required stereo camera drivers to publish the following topics:

- /stereo/left/image_raw
- /stereo/left/camera_info
- /stereo/right/image_raw
- /stereo/right/camera_info

Stereo adapter was developed to transform MiRo data and thereby making it compatible with *stereo_image_proc*. This is a standalone ROS package that was created exclusively for MiRo stereo vision purpose. It has following executable and corresponding nodes:

- scaleimage_left.cpp
- scaleimage_right.cpp
- subpub.cpp
- camera_info_publisher_left.py
- camera_info_publisher_left.py

Nodes *scaleimage_left* and *scaleimage_right*, subcribes to miro images and re-publishes after resizing it. The python scripts - *camera_info_publisher_left* and *camera_info_publisher_right* reads from .yaml files the parameters obtained as a result of Calibration and publishes CameraInfo message. The *subpub* node subscribes to the re-scaled images, CameraInfo topics and publishes it in the namespace as required by *stereo_image_proc*. It also broadcasts two transformations; one using odometry message between global map frame and MiRo body frame and the other from the eyelid frame to the frame of point cloud. MiRo developers do not guarantee synchronicity between the cameras and frame rates vary depending on the light received [10]. The subpub node successfully synchronizes the image and camerinfo topics subscribed by *stereo_image_proc* by editing the timestamp data of *std_msgs/Header* message type.

Odometry

The sensor information of MiRo is provided by the platform interface a customized message format *miro_msgs/platform_sensors* over the topic platform/sensors. The standard *nav_msgs/Odometry* messages needed to be published inorder to move the robot model for the visualization. A python script

shared by Yusha was modified to publish odometry messages. The same script also mapped the touch sensors of MiRo to *platform_control* topic. The head sensors of MiRo is used for rotation and the body sensors for translation.

Point cloud

ROS provides a message-based interface to efficiently communicate with the data types and structures provided by Point Cloud Library(PCL). The raw point cloud obtained contains exessive noise and density. Before using this data for various PCL algorithms, it is necessary to filter and downsample the point cloud. Example codes for the book - Learning ROS for Robotics Programming - 2nd Edition is shared online[11]. The nodes pcl_filter and pcl_downsampling were used to process the point cloud output of MiRo. The pcl_matching node was also modified for perform registration and matching using Iteratice Closest Point algorithm provided by PCL. But, due to the limited overlap region of stereo cameras in MiRo, an efficient 3-D map of the surrounding was not obtained.

Visualization

The *image_view* package was used for viewing an image topic. Disparity image published by *stereo_image_proc* package is viewed using the *stereo_view* tool. Stereo point cloud with RGB color published by *stereo_image_proc* is of message type *sensor_msgs/PointCloud2* and can be visualized using the RVIZ. Fig. 5 shows the very first Point cloud data viewed through RVIZ. It displays the 3-D reconstruction of phD student Syed Yusha (one of our project mentors). Notice the narrow range of the stereo data processed.

Robot Model of MiRo is provided in SDF XML format. Inorder include the robot model in RVIZ, the SDF file was converted to URDF format using python package *pysdf* [12]. The robot model contains 9 frames defined, including *miro_robot__miro_body__body* (body frame) and *miro_robot__miro_head__eyelid_lh* (left eyelid frame). The odometry data published from *subpub* node broadcasts a transformation between global frame *map* and *miro_robot__miro_body__body* frame. This results in mapping the motion of MiRo in real world with the robot model in RVIZ. The point cloud generated after filtering and downsampling is attached to the left eyelid frame. This fulfills the project objective in obtaining a 3-D representation of the surrounding environment of MiRo.

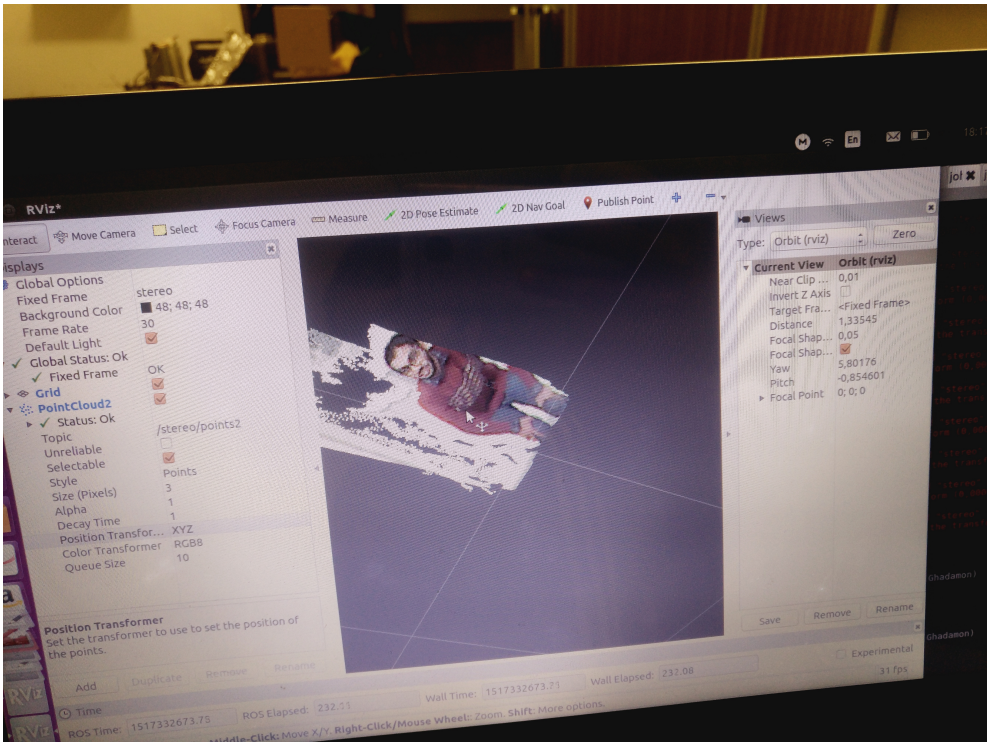

Figure 5: First Point cloud data obtained

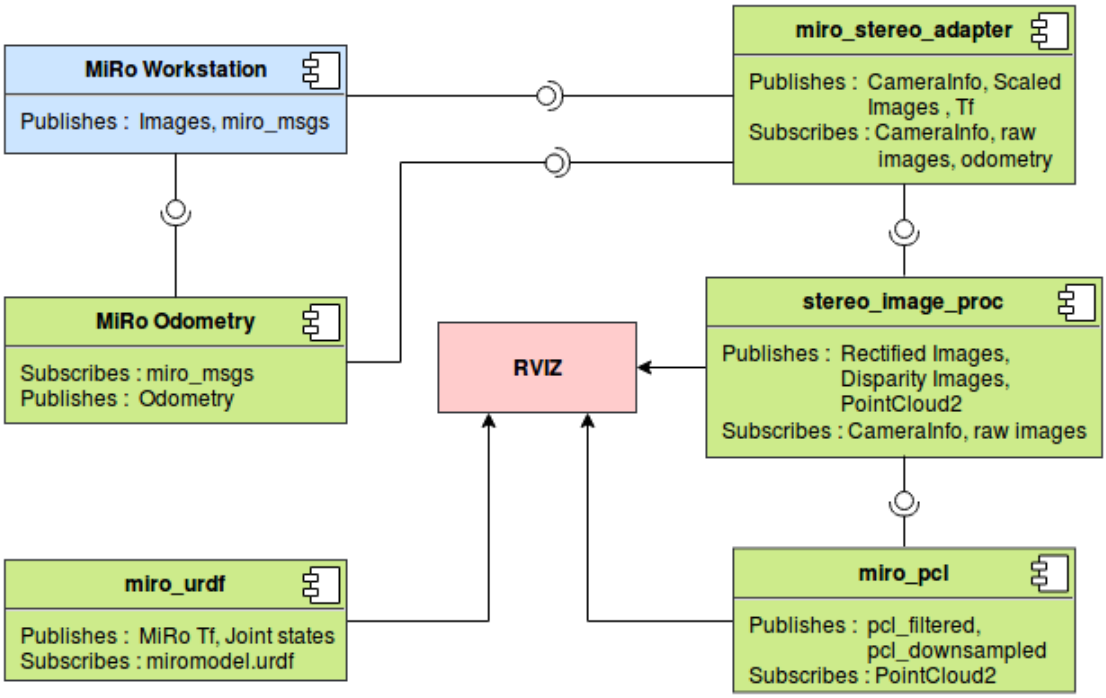

Figure 6: Schematic of Architecture used

Limitations of MiRo

- Narrow overlap region (about 30^0) of stereo camera.
- CameraInfo message not being published.
- Standard Odometry message not being published.
- Frame ID's not being published.
- Standard URDF file not being provided.

The above identified limitations remained a major hurdle in implementing any standard ROS based stereo-vision algorithm. The excitement and perseverance showed by the team resulted in taclkling these issues and successfully completing the objective. The infinite support received from Prof. Fulvio and mentors Luca and Yusha are deeply appreciated by the team. The team is also grateful for the help from other members of EmaroLab, including Kourosh.

Future Prospects

- Point cloud frame has been attached to the left eyelid frame of the robot model by trial and error to closely resemble the real world view. The package *ar_tracker_alvar* [13] can be used to calibrate a transformation from the eyelid frame of MiRo to the frame of point cloud. An initial attempt in the above approach failed due to the ar_tracker not being clearly detected through the point cloud. The detection of ar_tracker through the left image also posed some offset error that can be attributed to Camera Calibration.
- Generating 3-D map by stitching multiple point clouds using PCL-ROS interface or by using localization packages like RTAB-Map [14].
- SLAM

References

[1] http://consequentialrobotics.com/miro/,

[2] http://consequentialrobotics.com/domains/

[3] http://consequential.bitbucket.io/Technical_Interfaces_Platform_Interface.html

[4] http://consequential.bitbucket.io/Developer_Preparation_Prepare_workstation.html

[5] http://consequential.bitbucket.io/Developer_Preparation_Commission_MIRO.html

[6] http://wiki.ros.org/image_pipeline/

[7] http://wiki.ros.org/camera_calibration/Tutorials/StereoCalibration

[8] https://gist.github.com/rossbar/ebb282c3b73c41c1404123de6cea4771#file-yaml_to_camera_info_publisher-py

[9] http://wiki.ros.org/stereo_image_proc

[10] https://consequential.bitbucket.io/Technical_Sensors_Eyes.html

[11] https://github.com/AaronMR/Learning_ROS_for_Robotics_Programming_2nd_edition

[12] http://wiki.ros.org/pysdf

[13] http://wiki.ros.org/ar_track_alvar

[14] http://introlab.github.io/rtabmap/