Tampere University

Projection-based user interface Integration Tutorial

Preparation steps

- Hardware
- Preparing Software Environment
- Setting up Projectors/Kinect
- Installing Kinect drivers/ checking connection with ROS
- Installing Robot drivers, connecting robot to ROS

Hardware

- The following hardware is used in the setup
  - Workstation
    - Intel Core i7-6700HQ
    - 16GB RAM
    - NVIDIA GeForce GTX 970M
  - Depth Sensor
    - Microsoft KinectV2 (Other sensors are applicable, check compatib
  - DLP projector
    - BenQ MW550
  - Collaborative robot
    - UR5

Software Environment

- Installing linux
  - Recommended linux distribution is Ubuntu 18.04, since it is compatible with ROS melodic used in the setup. Installation instructions can be followed from here https://linuxtechlab.com/step-by-step-guide-to-install-ubuntu-18-04/
- Installing ROS
  - This implementation uses ROS Melodic distribution, Installation instructions can be followed from here http://wiki.ros.org/melodic/Installation/Ubuntu. Full-desktop installation is recommended
  - To check if ROS is installed correctly run **roscore** inside command line and see if it outputs any errors
- Download implementations for the Module's ROS nodes from https://github.com/Herrandy/HRC-TUNI/. The nodes should be downloaded to the src folder of your ROS workspace
- Build Module's nodes using the following commands
  - Source your default and workspace ROS environments
  - cd <workspace_folder>
  - rosdep install -r --from-paths .
  - catkin_make -DCMAKE_BUILD_TYPE="Release"

Installing Projectors and Depth Sensor

- Both depth sensor and projector are are installed above the workspace in close proximity to each other
- The devices should be installed perpendicular to the workspace area
- In case of projector, it is possible to install the device horizontally with the mirror. This can be used to increase the distance between the projector and the display plane and therefore increase the overall size of the projection.
- The devices can be installed either to the ceiling or with help of beam support
- The size of the projection is usually defined by the throw ratio **tr** of the projector, you can estimate the required height of the installation **h** for the given screen width **w** as **h=w*tr**

Kinect Drivers

- To use Kinect, one should first install device drivers
  - Kinect V2 https://github.com/OpenKinect/libfreenect2
  - Azure Kinect https://docs.microsoft.com/en-us/azure/kinect-dk/sensor-sdk-download
  - Check if device is working correctly by running k4aviewer for azure, or provided test scripts for KinectV2
- Next, install ROS interface for Kinect
  - Kinect V2 https://github.com/code-iai/iai_kinect2
  - Azure Kinect https://github.com/microsoft/Azure_Kinect_ROS_Driver
  - The drivers provide test launch files to check whether the node publishes correct data
- For other depth sensors please check compatibility with ROS

Setting up robot

- The module is designed to work with UR5 cobot
- The base of the robot is assumed to be stationary
- Install ROS drivers for the robot
  - ROS-industrial Universal Robot https://github.com/ros-industrial/universal_robot
  - UR modern driver https://github.com/ros-industrial/ur_modern_driver
- Check the IP address of your robot
- You can check if the robot connection is working by using **ping <robot_ip>** command
- To check driver, run **roslaunch ur_modern_driver ur5_bringup.launch robot_ip:=<robot_ip>.** Check if /joint_states topic publishes joint data

Projector Interface

- Robot-Projector calibration
- Safety border
- Setting up UI
- Setting additional Parameters
- Testing Border
- Testing UI

Purpose of the module

- This module provides a visualization for the depth-based safety system and the UI using a projector/RGBD sensor setup
- The safety area is visualized as border around the robot. The border is generated dynamically around the robot during its movement
- The module also visualizes UI elements that can provide instructions for the operator and can be interacted with through RGBD sensor

Robot-Projector Calibration

- Finding transformation between robot coordinate system and the projector is similar to robot-camera calibration. Unfortunately explicit script for this calibration is not included in package, so procedure would need to be performed manually. The steps of the procedure would be following
- Perform TCP calibration (or use TCP from Robot-camera transform if possible)
- Run projector and project some pattern with some points of interest, image coordinates of which are known (in simplest case points of interest can be the corners of the projector image. The coordinates of the points therefore would be (0,0),(width,0),(0,height),(width,height) where width and height define resolution of the image projected). At least four points are needed.
- Align TCP and points of interest and record XY coordinate of the TCP point and corresponding coordinate of the point in the image (x_rob,y_rob) (x_im,y_im)
- After points are collected you can use OpenCV library to find the homography matrix of the transformation. The operation can be done with **findHomography** function. Example of how homography is computed can be checked at https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html
- Copy the homography matrix to **robot_projector_homography.txt** file located inside the **unity_msg/configs**

Safety border

- For visualization of the safety border, the ‘Depth-sensor safety model for HRC’ module should be installed
- To install the module, please check the corresponding tutorial

Setting up UI

- It is assumed that the steps for safety module were performed, since some parameters are shared between modules
- Projector node of the module contains a **place_interface_components.py** script, which can be used to set up the elements of the UI in the projection plane
- Names/Paths to the elements should be defined inside the script, for example the elements used in the demo are stored in unity_msgs/config/mobile_demo folder
- The script can be run using **rosrun projector place_interface_components.py** command
- Position can be set using arrow keys, ‘r’ rotates pattern 90 degrees, ‘b’ and ‘m’ changes scale, ‘m’ switches to a different pattern, ‘p’ saves current pattern location, ‘l’ saves all patterns
- The position of the UI elements should be also defined in the space of the camera image. The definition of buttons is located inside handle_interaction_markers.py script, lines 49-53
  - No specific script exist right now to define this positioning, here’s one way to define it
    - Run **rosrun projector projector_interface.py** to display image of UI to the projector
    - Extract image published on **/kinect2/sd/image_color_rect** publisher
    - Manually find center coordinates and radius of the buttons in the image
    - Set coordinates in the script to the corresponding buttons

Setting additional parameters

- Some parameters may need to be adjusted for module to work properly
- The parameters are stored inside **unity_msgs/configs/config.yaml** file
  - interaction_button_thres – the threshold in depth that defines whether the button was interacted with

Testing safety border visualization

- Run your test program for the robot
- Run the following commands in separate consoles
  - **roscore**
  - **roslaunch ur_modern_driver ur5_bringup.launch robot_ip:=<robot_ip>**
  - **roslaunch kinect2_bridge kinect2_bridge.launch max_depth:=2.0 publish_tf:=true**
  - **rosrun projector projector_interface.py**
  - **rosrun projector handle_interaction_markers.py**
  - **rosrun robot dashboard_client.py**
  - **roslaunch safety_model detect.launch safety_map_scale:=100 cluster_tolerance:=0.005 min_cluster_size:=200 anomalies_threshold:=20 cloud_diff_threshold:=0.02 viz:=false**
- You should see red border around the robot

Testing UI

- Run same commands as in previous step
- When putting your hand over ‘start’ and ‘stop’ buttons, the UI should change
- Putting your hands over ‘stop’ and ‘dead man switch’ buttons should stop the robot
- Putting your hands over ‘start’ and ‘dead man switch’ buttons should start robot again
- **dashboard_client.py** console should output what buttons are being interacted with.
- If buttons do not respond, try positioning your hand at higher locations over the button or changing **interaction_button_thres** to lower values

Config files

- List of important configuration files for the modules
  - **/calibration/launch/tf_broadcast.launch** –transformation from Kinect to robot
  - **/unity_msgs/configs/config.yaml** – parameters for the safety and UI modules
  - **/unity_msgs/configs/robot_projector_homography.txt** – transformation from robot to projector space
  - **/unity_msgs/configs/mobile_demo/projector_buttons.yaml** – locations of the UI elements on the projector plane

Maintenance and troubleshooting

- Here we give general recommendations for troubleshooting, for more specific problems please email to **dmitrii.monakhov@tuni.fi**, we will help
- General tips for troubleshooting
  - Always check if you sourced ROS environment on any new console tab before running commands. If you are using a single ROS environment, it may be easier to add source commands to **.bashrc** so it is done automatically
  - Check output of the scripts in the console, as it can give hints for errors
  - Remember that Time-of-flight depth sensors have a warm-up period during which temperature of the sensor and processed values can drift. This can cause very unpredictable and hard to catch errors, so we recommend running the sensor for 30-60 minutes before the test
  - Don’t forget to rebuild the modules after making changes in config/source file

Tampere University

Thank You

Dmitrii Monakhov
Dmitrii.Monakhov@tuni.fi