Tampere University

Depth-based safety system Integration Tutorial

Tampere University

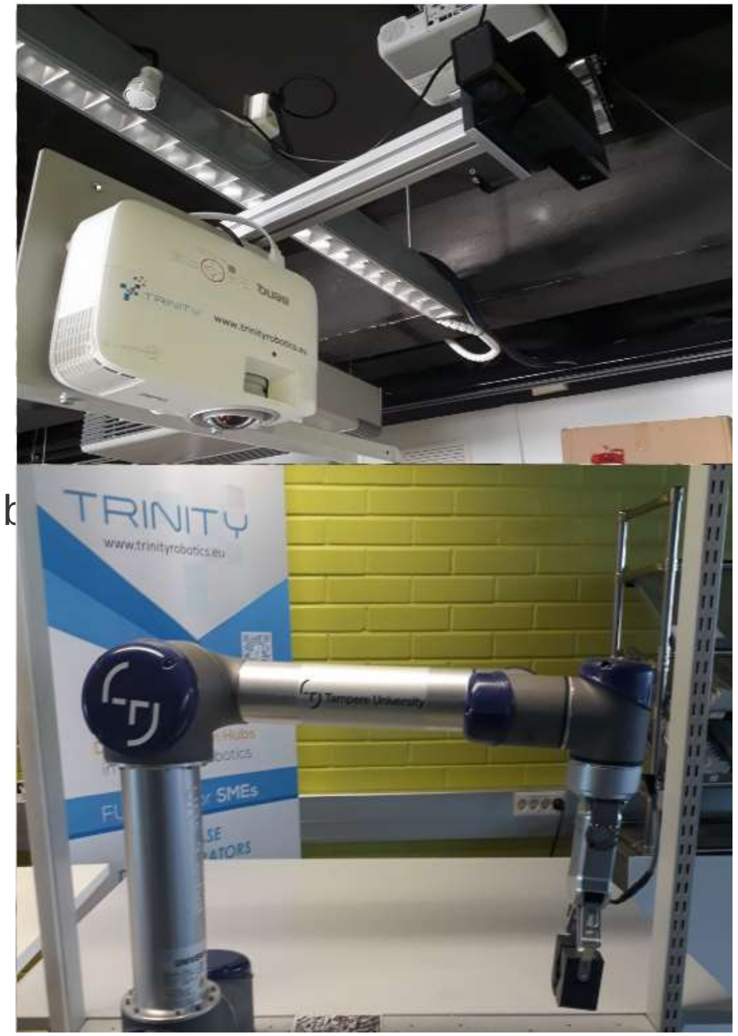
Preparation steps

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

Tampere University

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

Tampere University

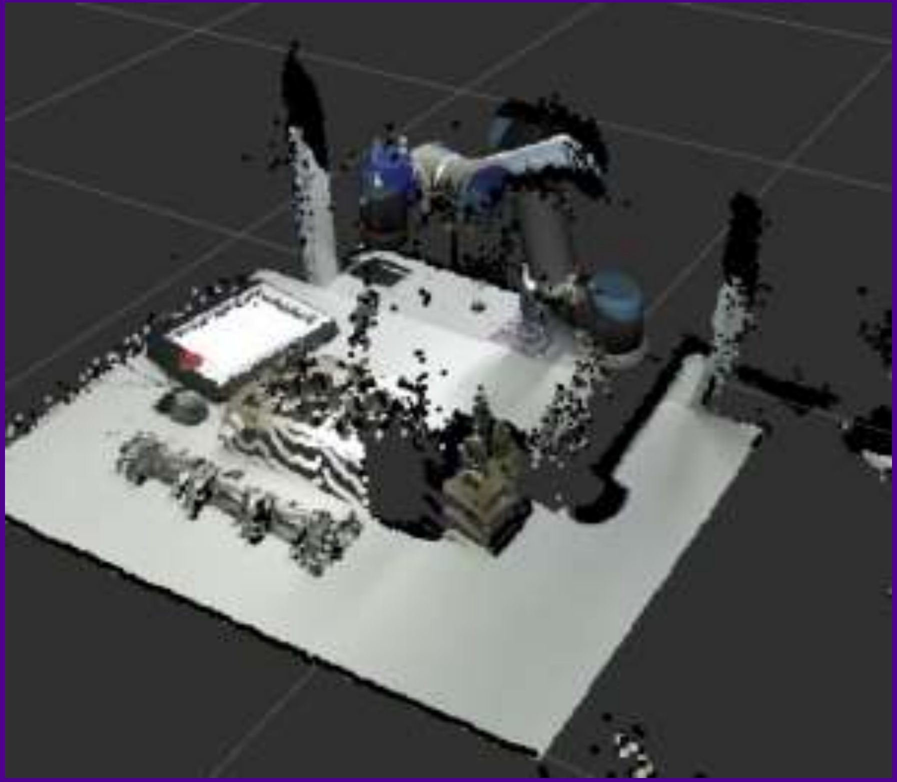
Setting up robot

- 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

Tampere University

Depth-based safety system

- TCP calibration
- Finding transformation between Kinect point cloud and joint positions
- Projector-robot transformation
- Setting parameters of the safety border
- Testing



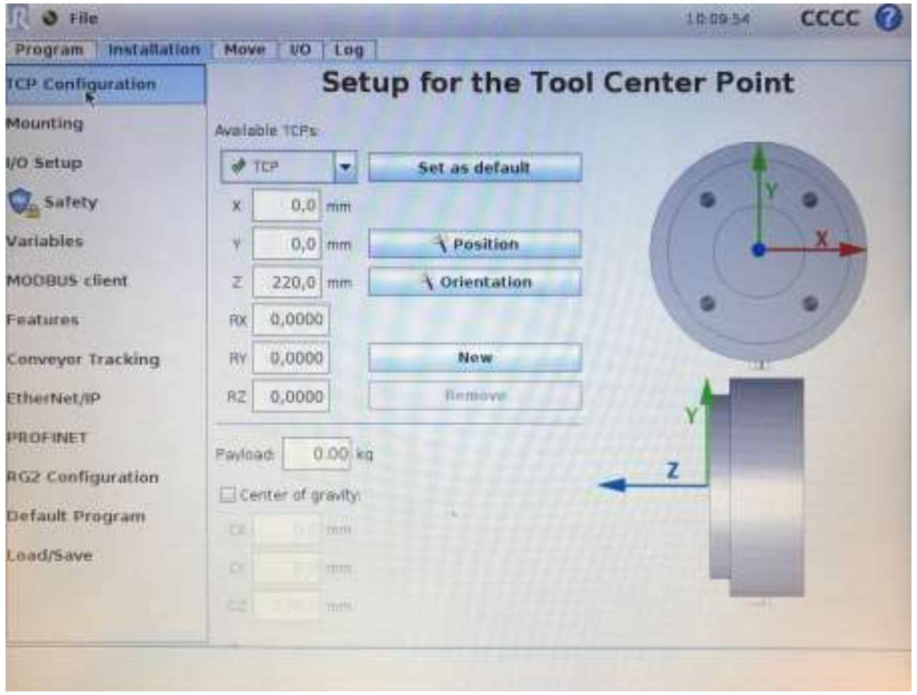
Tampere University

Purpose of the module

- This module provides a safety model for collaborative robots using depth sensor setup
- The safety system divides space into robot zone and operator zone
- If a change in depth happens at the border between the zones, the robots stops its program until violation is resolved

TCP calibration

- Calibration between Kinect point cloud and robot position is done by detecting a red dot marker and finding the transformation of the marker coordinates between Kinect and robot coordinate systems
- Before performing the calibration, the user need to find the offset between the red dot marker center and the robot end effector (unless the marker is attached directly to the center of the flange). This can be done using touch point method, with help of the following library https://github.com/Jmeyer1292/tool_point_calibration
- Four points or more are recommended for accurate estimation
- The calculated offset should be inserted into **calibration_kinect2robot.launch** file inside launch folder of the calibration module



Setup for the Tool Center Point

Available TCPs: TCP

X 0,0 mm
Y 0,0 mm
Z 220,0 mm
RX 0,0000
RY 0,0000
RZ 0,0000

Payload: 0.00 kg

☐ Center of gravity:

Robot-camera Calibration

- After TCP calibration is done, install red marker at the robot TCP point such that it would be visible from the Kinect
- Run roscore, and run the following script in another console
  - **roslaunch calibration calibration_kinect2robot.launch**
- Move the robot to some position visible in the camera space and press **c** to store the coordinates of the point
  - At least three points is needed, more points would give better estimation
  - The points should not lie on the same line
- After enough points were collected, press s to calculate the transform
- Copy the transformation into **tf_broadcast.launch**
  - First three values should be translation, 4 subsequent ones - rotation
- After values were copied, you can test the calibration with the following commands
  - **roslaunch ur_modern_driver ur5_bringup.launch robot_ip:=<robot_ip>**
  - **roslaunch kinect2_bridge kinect2_bridge.launch publish_tf:=true**
  - **roslaunch calibration test_calibration.launch**
- The commands should run RViZ, in which the robot model and point cloud should align

Setting other parameters

- There are several parameters for the safety border which may need to be adjusted
- The parameters are stored inside **unity_msgs/configs/config.yaml** file
  - dynamic_workspace_size – defines the radius of projector border against the robot
  - safety_area_offset – defines how wide the visualized border line would be
  - Sensitivity Parameters – change if safety stop is too/not enough sensitive
    - cluster_tolerance – spatial cluster tolerance in Eucledian clustering for the module (lower value – less sensitive)
    - min_cluster_size – minimum number of points for defining a cluster (less value – more sensitive)
    - cloud_diff_thresh – minimum change in two depth pixels to be considered as change
    - nearest_object_threshold – minimum change in depth to be considered a violation
  - workspace_limits – the limiting corners of the workspace zone

Testing Safety module

- To test module, you can write a simple robot program that would move robot around
- After you tested that your program works, run the robot program on loop and execute the following commands in the console
  - **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 robot dashboard_client.py**
  - **roslaunch safety_model detect.launch safety_map_scale:=100 cluster_tolerance:=0.005 min_cluster_size:=200 viz:=false cloud_diff_threshold:=0.02**
- The robot should stop when you move too close to it

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 configuration

Tampere University

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