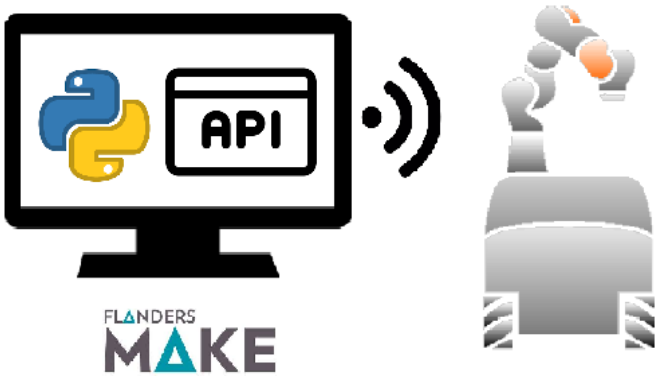
KMR External Control Module

Developer version



Authored by:

Ali Bin Junaid

Robotics Application Engineer

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

Table of Contents

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

1. Training Module Overview

This training module will provide a technical overview on the 'KMR External Control Module'. This training will cover the following topics:

- KUKA KMR iiwa robot Overview
- System Architecture Overview
- Components Overview
- Programming & Motion Class Implementation
- External Control Interface

The module provides an interface for easy programming of KMR iiwa. This interface works on top of the native software utilizing the autonomous functionalities of the KMR iiwa. Furthermore, motion class has been implemented in the internal controller of KMR iiwa, with primary navigation functions used in a typical application available and parametrized. Using wireless communication over UDP, communication interface implemented in this module establishes bi-directional communication between external control PC and KMR iiwa internal controller. The interface allows intuitive programming of KMR iiwa making modification and reprogramming of automated applications possible with decreased time and effort. Utilizing autonomous functionalities of KUKA Navigation Software, the proposed interface provides higher level programming blocks to be used, hence fundamentally reducing the application (re-)programming effort.

2. KUKA KMR iiwa robot Overview

KUKA KMR iiwa platform (Figure 1) is a combination of LBR iiwa robot and an omnidirectional mobile robot KMP 200 omniMove with high degree of flexibility and mobility. Programmed on its native software, KMR iiwa has advanced features for mapping, localization, trajectory generation and robot control allowing complete autonomous operation with high accuracy. "KUKA.NavigationSolution" is KUKA's navigation solution for autonomously navigating vehicles. Whether navigation, control, management or monitoring: navigation solution covers all mobility requirements. Using the SLAM method – which is an acronym for Simultaneous Localization And Mapping – the platform is able to pinpoint its location in real time on a map of its environment created from the data of the safety laser scanners and wheel sensors. The KUKA Sunrise.OS system software is used for the KUKA KMR iiwa. This software is tailor-made to operate lightweight robots and offers functions for programming, planning and configuring lightweight robot applications.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

Figure 1: KUKA KMR iiwa Robot

3. System Architecture Overview

3.1. Overview of System Structure

Figure 2 illustrates the basic configuration and interrelationship of the software components of the KMP/KMR robot system.

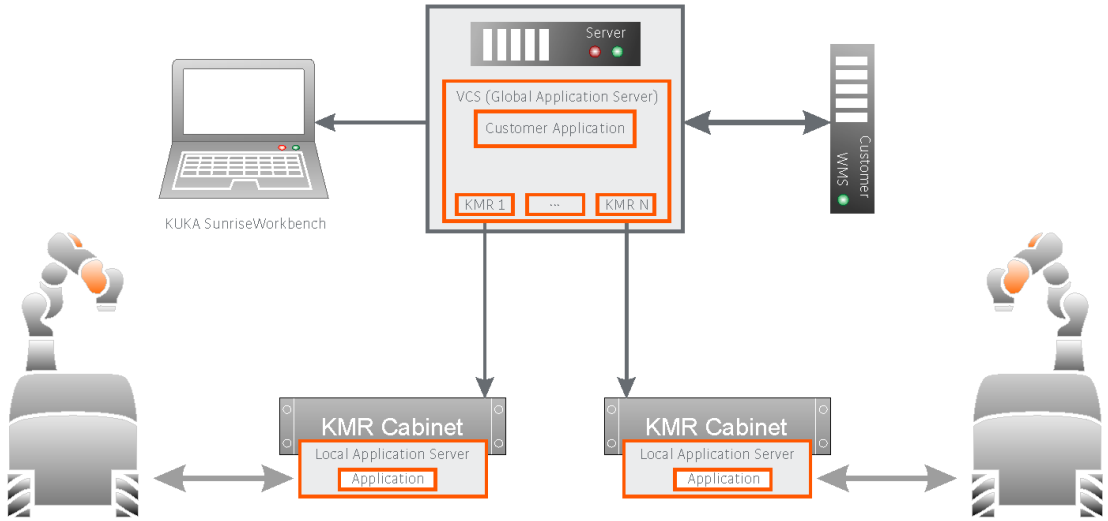


Figure 2: Description of System Architecture

Programming and configuration of the Robot is done with Sunrise Workbench. Navigation server controls the applications and vehicles. One important software component is the FleetManager, which takes control of the vehicles with regard to task planning. The navigation server can take over the controller using an external computer (multiple vehicles) or a vehicle (single vehicle) and management and control of multiple vehicles via the navigation server (KMR_1 ... KMR_N). On the individual vehicles, a local application server monitors and executes vehicle programs. This server runs on the control computer of the robot (KMR controller).

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

3.2. Overview of Project Structure

Different projects are required for controlling the vehicles. Depending on the number of vehicles to be controlled, these projects can be configured on different systems.

3.2.1. Single Robot

When single robot is used, all projects and applications on the robot itself are synchronized and configured in a following manner:

- Project for vehicle control on the NavBox. Alternatively, this project can also be configured on an external server.
- Tasks for the controller of the LBR iiwa or other external devices are configured on the Sunrise controller.

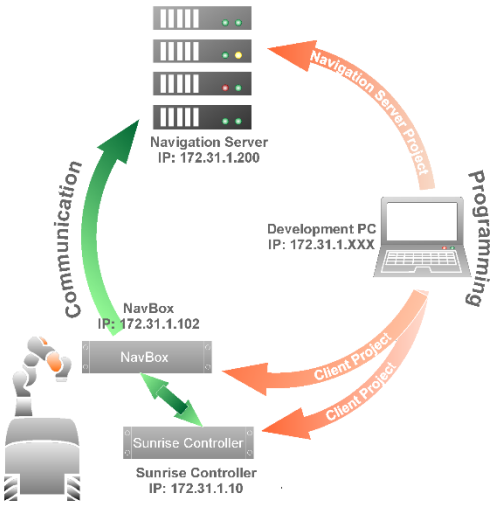


Figure 3: Overview of Project Structure

3.2.2. Multiple Robots

When multiple robot is used, projects and applications are synchorinzed and configured in a following manner:

- Projects for vehicle control should be implemented on an external server.
- Tasks for the controller of the LBR iiwa or other external devices are configured on the relevant Sunrise controller.

3.3. Overview of Commanding Structure

Server applications are synchronized on the navigation server. The navigation server forwards the tasks to the individual vehicles (Navigation Box) via the software package FleetManager. The navigation client forwards the motion requirements to the Sunrise controller. From there, the requirements are calculated and executed (Sunrise Mobility). Furthermore, applications destined for the LBR iiwa or other devices can also be processed.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

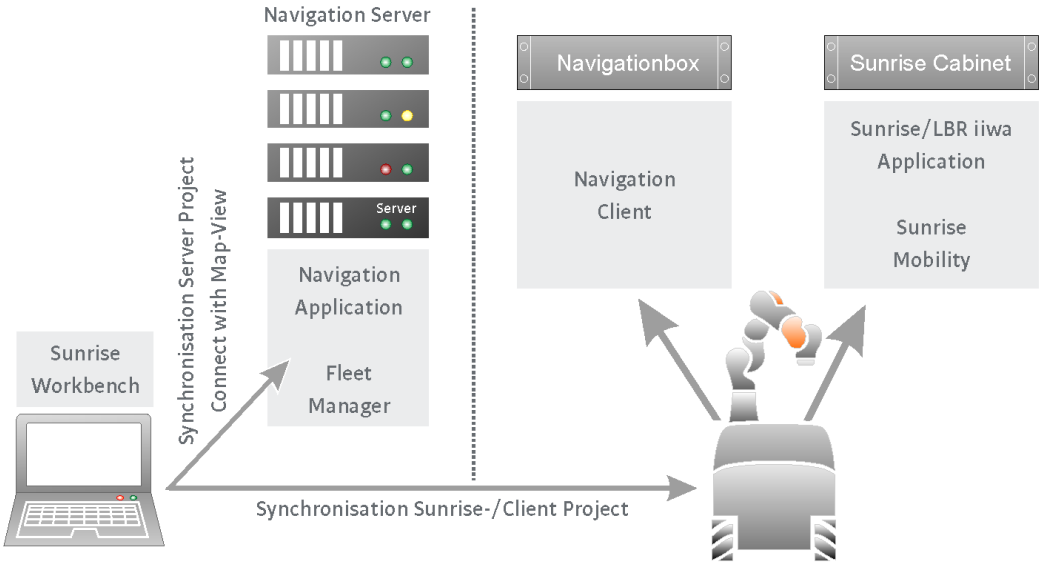


Figure 4: Overview of Commanding Structure

4. Components Overview

4.1. KUKA Navigation Solution

The KUKA Navigation Solution software is used for autonomous navigation, control, management and monitoring of mobile platforms.

Central functions of KUKA Navigation Solution:

- Offline creation and configuration of navigation projects
- Project transfer to the control computer of the navigation system
- Online detail configuration of navigation projects
- Online control of mobile platforms via the navigation server
- Programming of applications

Computers in the navigation system:

- **Work computer:** The KUKA Navigation Solution software is installed on this computer.
- **Control computer:** The KUKA Navigation Solution software installs the navigation server with the navigation project on this computer. The control computer communicates with all mobile platforms of the project via WLAN. All the mobile robot platform functionalities are programmable in this computer.
- **KUKA NavBox:** This computer is integrated into the mobile platform. It serves as an interface between the control computer and the mobile platform.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

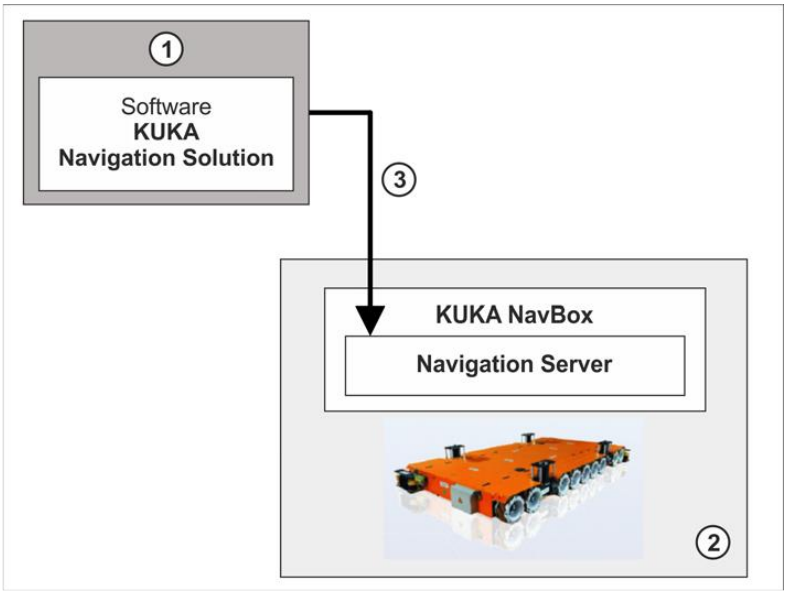


Figure 5: Typical Standalone Application Architecture

If a project only contains a single mobile platform, no separate control computer is required. The navigation server can be installed directly on the KUKA NavBox.

4.2. Installing Navigation Solution

Installation instructions can be found by referring to Section 4 of KUKA's Navigation Solution Manual (version 1.14 en)*.

*Navigation solution manual is provided by the KUKA vendor with the purchase of the robot.

4.3. Overview of General User Interface

The user interface of KUKA Navigation Solution consists of several views. The combination of several views is called a perspective. Navigation Solution offers various preconfigured perspectives. Perspectives can be activated and deactivated. The default perspective is Programming. Another important perspective is the KUKA Map Perspective.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

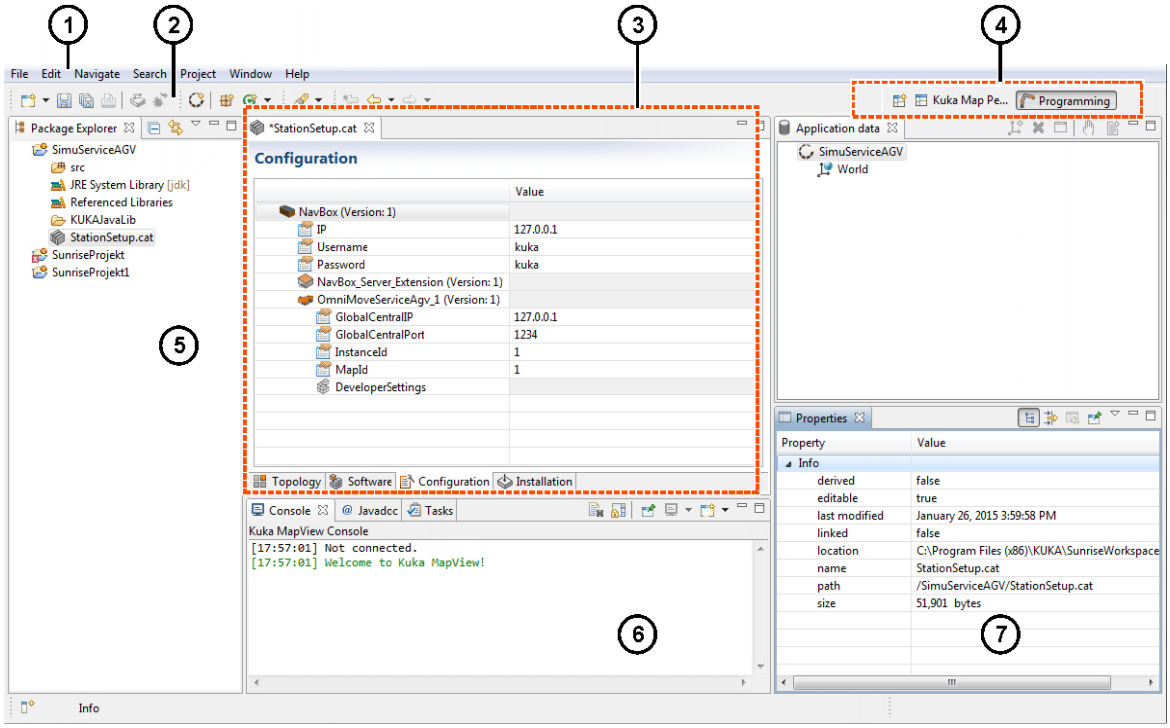


Figure 6: User interface with Programming perspective

Table 1: User inteface items description

Item	Description
1	Menu bar
2	Toolbars
3	Editor area Opened files are displayed here.
4	Perspective selection
5	Package Explorer view This view contains the projects created and their corresponding files.
6	Tasks, Console and Javadoc views Tasks: Displays tasks which a user has created Console: Information output during the runtime of an application Javadoc: Displays
7	Properties view If an object is selected in a view, its properties are displayed here.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

5. Programming and Motion class Implementation

5.1. Basic startup and programming KMR

To get started, KUKA's **"Training KMP/KMR – Start-Up and Programming KUKA Navigation Solution Manual*"** will step by step guide to get the familiarization with the platform and will enable the programmer to get acquainted with basic functionalities of the robot. Extensive robot application functionalities using programming and exercises are available. These basic functionalities include:

- Setting up software for the application
- Generating and using Map for autonomous operation
- Programming Basics

*Training KMP/KMR – Start-Up and Programming KUKA Navigation Solution manual is provided by the KUKA vendor with the purchase of the robot. Tasks for the LBR iiwa manipulator can be controlled using remote task commanding. Please refer to section 6.8 of the manual for more details.

5.2. Motion Class

FlandersMake has developed *Motion Class* which makes it easier to program applications on KMR quickly and flexibly.

The primary functions used in a typical application are as follows:

- Get current location of the robot on the map
- Execute virtual line motion to the location given
- Execute fine localization on the location having fine localization data
- Execute Graph Motion to goal node from the current node

The motion class incorporates these functionalities which can be executed by just passing the desired parameters to the class method.

5.2.1. Get current location

The below method in the *Motion class* allows to get the current location of the robot to know on which location robot is currently at. The function gets the current pose of the robot and compares it one by one with the pose of the specified locations and returns the location matched if any.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

```
public int getcurrentloc(LocationData _locDat, MobileRobot _rob) {
                int robotloc = 0;
                int currentlocid = 0;
                Location currentloc = _locDat.get(currentlocid);
                for (int i = 39; i < 45; i++) {
                        currentlocid = i;
                        currentloc = _locDat.get(currentlocid);
                        double differencex = Math.abs(_rob.getPose().getX()
                                        - currentloc.getPose().getX());
                        double differencey = Math.abs(_rob.getPose().getY()
                                        - currentloc.getPose().getY());
                        if (differencex < 0.15 && differencey < 0.15) {
                                robotloc = currentlocid;
                                break;
                        } else {
                                robotloc = 0;
                        }
                }
                return robotloc;
        }
```

5.2.2. Execute Goto Location

The below method in the *Motion class* allows to pass the destination location and robot will execute a virtual line motion to go to the desired location.

```
public void executegoto(LocationData _locDat, MobileRobot _rob, int gotoloc) {

                Location Loc = _locDat.get(gotoloc);
                // move robot to the taught location
                _rob.execute(new VirtualLineMotion(_rob.getPose(), Loc.getPose())
                                .setMaxVelocity(0.4).setMaxAcceleration(0.3)
                                .setMaxDeceleration(0.3));
        }
```

5.2.3. Execute Graph Motion

The below method in the *Motion class* allows to execute a Graph motion to go to the goal node from the current node.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

```
public void executeGraphMo(GraphData _graDat,int currentloc, int graphID, int nodeID, int
goalnodeID, FleetManager _fleetMan, MobileRobot _rob) throws LockException, InterruptedException{

                TopologyGraph graph = _graDat.get(graphID);

                TopologyNode node = graph.getNode(nodeID);
                // set robot to graph
                ChangeGraphCommand graCom = new ChangeGraphCommand(graphID, nodeID);
                _rob.execute(graCom);
                // _fleetManager needs the lock of the robot
                _rob.lock();

                // move on the graph
                TopologyNode goalNode = graph.getNode(goalnodeID);
                GraphMotion graMo = new GraphMotion(graph, goalNode);
                _fleetMan.execute(graMo);

                // fleetMan releases the rob after the motion ==> get back the lock
                _rob.unlock();

                // (Optional)  the fleetMan needs to know, that the robot moves now to another pose.
// Otherwise the node is blocked for other AGVs
                 _rob.execute(new RemoveFromGraphCommand());
                 _rob.execute(new RelativeMotion(1, 0, 0.1));

        }
```

IMPORTANT PREREQUISITES:
Following Prerequisites are required for the graph motion to be executed.

- Graph, nodes and edges created, and parameters are set (Please refer to the documentation reference from Section 5.1 on how to create)
- Robot is set to the graph and node using Sunrise Map Perspective
- Starting node should be the one where robot is currently at

5.2.4. Execute Fine Localization

In many applications, the exact position of the robot is necessary for the process. For this, the exact position of the robot can be determined relative to a specified taught location. The scanning information obtained by the laser scanners from the environment is used to calculate the exact position. The more features there are to find in the environment, the greater the quality of the fine localization. Furthermore, a fixed position must also be known. The reference point is the centre point of the vehicle or the vehicle coordinate system.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

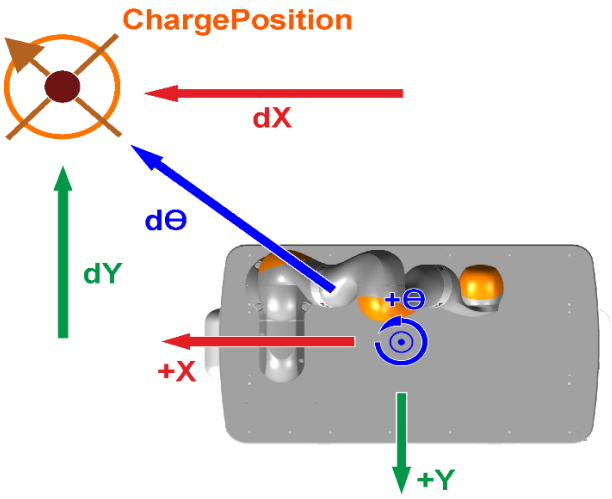


Figure 7:Description of Fine Localization

The below method in the *Motion class* allows to execute a Fine Localization motion on the location robot is currently at. This function executes very fine movement to get as close as possible to the location. Please refer to Section 6.7 of the Manual referred in Section 5.1 of this document.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

```
public void executeFineLoc(LocationData _locDat, int fineloc) {
		int i = 0;
		Location taughtLoc = _locDat.get(fineloc);
		if (taughtLoc.hasSensorData()) {
			do {
				// do the fineLocalization
				FineLocalizationRequest finLocReq = new FineLocalizationRequest(
						taughtLoc);
				FineLocalizationContainer container = _rob.execute(finLocReq);

				// extract the relative Pose from the location to the center
				// of the
				// kmp and invert it
				_offset = container.getRobotPose().invert();

				if (_offset.norm() <= 0.005
						&& Math.abs(_offset.getTheta()) <=
Math.toRadians(0.5))
					break;

				// move about the offset
				_rob.execute(new RelativeMotion(_offset).setMaxVelocity(0.1));
				i++;
				// } while (true);
			} while (i < 1);

		} else {
			_log.error("No sensordata!");
		}

	}
```

5.3. Sample program using Motion Class

Below you can find the code snippet of an example program which utilizes motion class to perform basic motion functions of getting current location of the robot and executing graph motion to go to a specific node. Note that how using motion class simplifies the programming of the robot for motion functionalities.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

```
import funtionclasses.Motion;

@NavTaskCategory
public class ClassTest extends RoboticsAPITask {
	// Declaration of variables
	@Inject
	private MobileRobotManager _robMan;
	private MobileRobot _rob;
	@Inject
	private ITaskLogger _log;
	private int _robotId;
	@Inject
	private GraphData _graDat;
	@Inject
	private FleetManager _fleetMan;
	@Inject
	private LocationData _locDat;
	private int graphId;
	@Override
	public void initialize() throws Exception {
		_robotId = 1;
		_rob = _robMan.getRobot(_robotId);
		_log.info("Initialize finished.");
		graphId = 9;
	}
	@Override
	public void run() throws Exception {
		_log.info("Starting application...");
		try {
			_rob.lock();
			int currentlocation=0;
			Motion robotfns = new Motion();
			currentlocation = robotfns.getcurrentloc(_locDat, _rob);
			robotfns.executeGraphMo(_graDat, currentlocation, graphId, 1, 4, _fleetMan, _rob);
			currentlocation = robotfns.getcurrentloc(_locDat, _rob);
			_log.info("My location is "+ currentlocation);
			if(_locDat.get(currentlocation).hasSensorData()){
				_log.info("Location has Fine Localization Data. Fine Localizing");
			}
			else{
				_log.error("Location has no data");
			}
		}
		catch (LockException e) {
			_log.error("Already locked.", e); // thrown, when the locking failed
		} catch (InterruptedException e) {
			_log.error("Interrupted.", e);
		} finally {
			_rob.unlock();
		}
		_log.info("Application finished.");

	}
	@Override
	public void dispose() throws Exception {
		_log.info("Dispose finished.");
	}
}
```


Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

6. External Control Interface

6.1. Overview of External Control Interface

Network communication via UDP and TCP/IP is possible with the robot as certain ports are enabled on the robot controller for communication with external devices via UDP or TCP/IP. The following port numbers (client or server socket) can be used in a robot application: 30,000 to 30,010. This basically means that you can create a simple UDP Server on the robot which can listen to the messages sent by the external controller on the specific port. Importantly, UDP message structure in Table 2 can be customized to the specific needs giving flexibility for the communication. Moreover, you can connect using WLAN/LAN from any PC opening up specific port (within the range allowed i.e. 30,000 - 30,010) and you can start sending/receiving UDP packets.

To setup UDP interface, basic socket programming methods in JAVA are used to establish bi-directional communication. Several examples and tutorials are freely available online. (Example: "A Guide To UDP In Java", https://www.baeldung.com/udp-in-java)

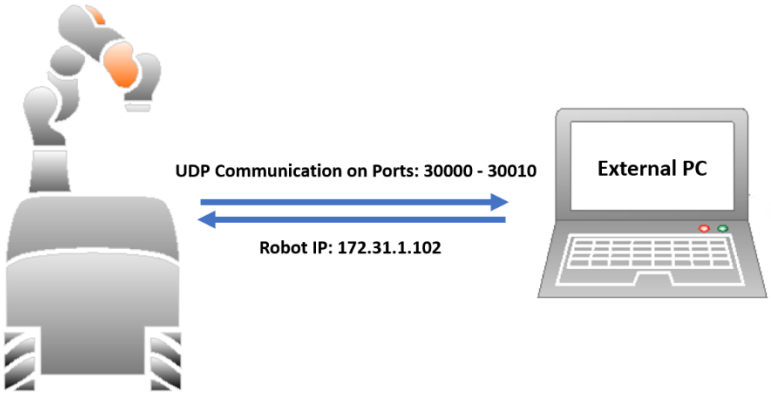


Figure 8: Basic UDP Communication Architecture

6.2. UDP message format for motion class functionalities

Table 2: UDP message structure format

Field Name	Size (bytes)	Description
Length	2	Length of the entire UDP datagram including header (Length+Type)
Type	2	Type of the function
Data	Variable	Data containing parameters of the function

Table 3 provides the executable functions of the motion class structured with the respect to type of function and their parameters. With the implementation of basic UDP client/server application, different methods can be created which processes the UDP messages (datapackets)

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

according to the format in Table 2 and calls the respective function from the motion class. External communication server/client can be implemented in any language. This module was tested in the UDP client interface implemented in Python3.

Table 3: Motion Class functionalities for external control

Type	Service type	Parameters	Description	Data Type	Response
Goto Location	*ExecuteGoto*	gotoloc	Location to Go	Int	"Done"
		vel	Velocity	Double	
		acc	Acceleration	Double	
Goto Node on Graph	*ExecuteGotoGraphMo*	graphID	Graph ID	Int	"Done"
		Currentloc	Current Location	Int	
		nodeID	Current Node ID	Int	
		goalID	Goal Node ID	Int	
*Manipulator Task	*ExecuteManipulator*	Taskname	Task to be executed	String	"Done"
Fine Localization	*ExecuteFineLoc*	fineLoc	Taught Location with defined fine localization parameters	Int	"Done"
		tries	# of tries	Int	
Get Current Location	*ExecuteGetCurrentLoc*	minLoc	Lowest ID # in Map Data	Int	""+integer (Int = Current Location ID)
		maxLoc	Highest ID # in Map Data	Int	
ShutDown	*ExecuteShutDown*	Not Applicable			"Connection Closed"
Initialize	*ExecuteInit*	Not Applicable			"Connection Intialized"

**Manipulator task is not added by default in the motion class provided by FlandersMake.*

Please note that the UDP message structure in Table 2 and function types in Table 3 are provided as a reference for implementation of motion class for external control. The UDP message structure and format can be customized as per requirements.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.

For more information or support regarding this training module, contact Trinity Project Partners.

Trinity project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825196.