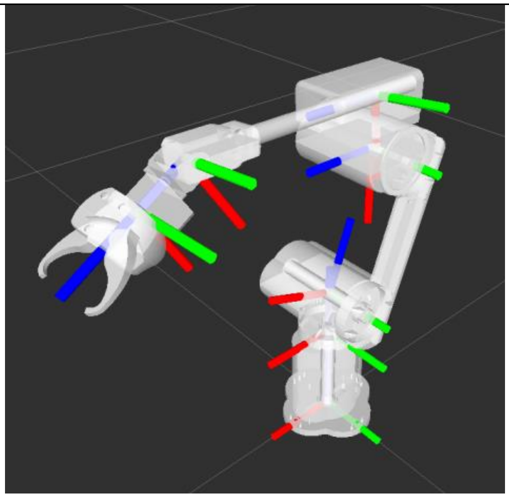
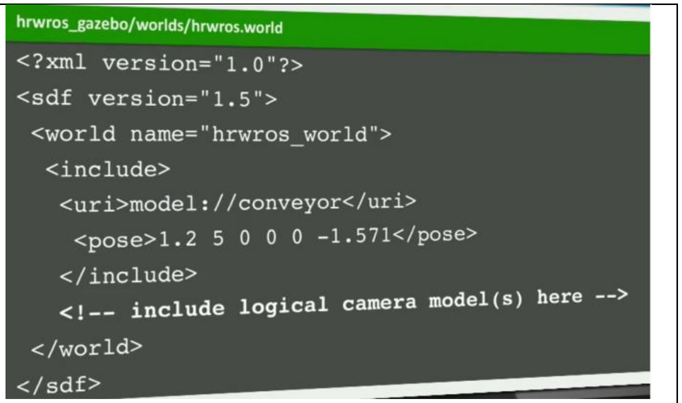
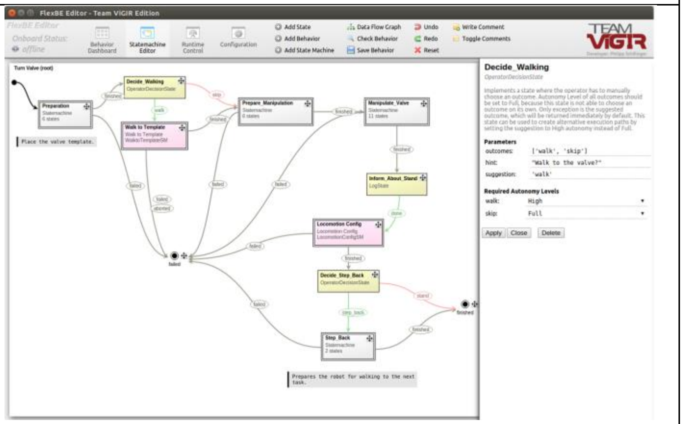
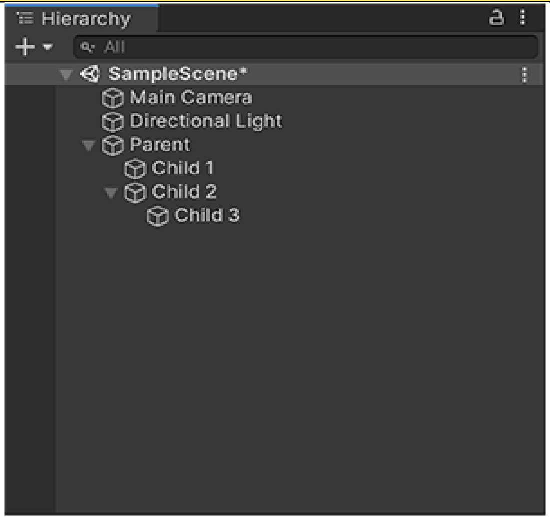
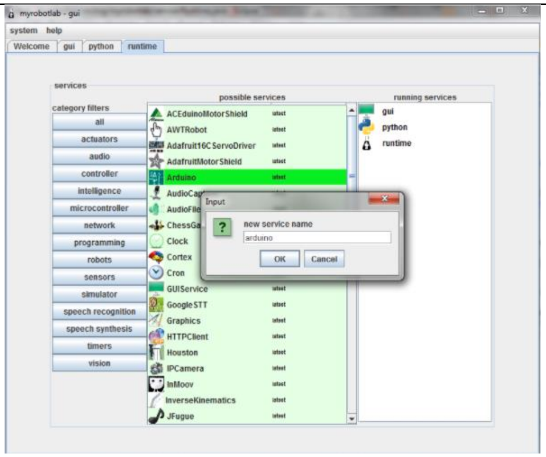
Intro to Robotics software packages

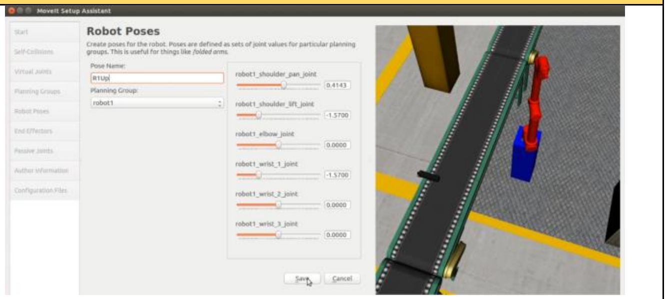
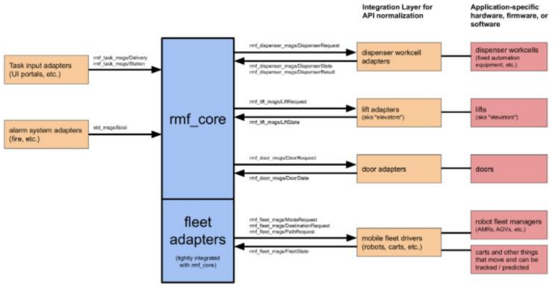
The current document summarizes some of the ROS-based and non-ROS based software, packages, and filetypes used in robotics research. Based on their application, the software are categorized into four groups control, mapping, physics, and simplification.

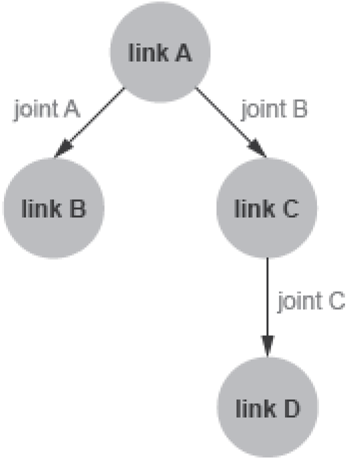
ROS BASED SOFTWARE				
Software	Description	Language	Applications	SCHEMATIC
ROS	Robotics Operating System: composed of nodes and topics (edges)	Python/C++	Control: The directory structure of packages. Can be used with all software in this section.	ROS = + + + plumbing tools capabilities community Image Ref
RViz	A 3D visualization tool. It shows what the robot thinks is happening: Visual simulation of the digital environment of the robot in a way that is understandable in human terms	Graphical User Interface	Mapping: Renders virtual environment; used to tweak robot sensors	 Image Ref

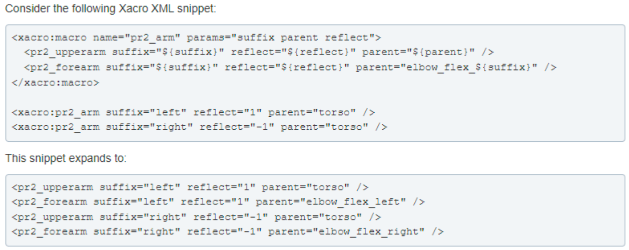
Gazebo (Ignition)	A 3D simulator. It shows what is happening to the robot: Defines a virtual world for a simulation robot. A gazebo world file is composed of many .sdf models. ROS support is directly built into Gazebo	Graphical User Interface, Python (for math)	Mapping:
			Renders virtual environment. Can use .sdf and .urdf models together via spawner node, but the environment is not as clear to humans as it is in RViz
			Physics:
			Uses tweakable physics which can affect robot movement
				
				hrwros_gazebo/worlds/hrwros.world
				`<?xml version="1.0"?>`
				`<sdf version="1.5">`
				`  <world name="hrwros_world">`
				`    <include>`
				`      <uri>model://conveyor</uri>`
				`        <pose>1.2 5 0 0 0 -1.571</pose>`
				`      </include>`
				`      <!-- include logical camera model(s) here -->`
				`  </world>`
				`</sdf>`
				Image Ref

FlexBE	A way to implement state machines within ROS for behavior design	Graphical User Interface	Simplification: Allows the user to use a non-coding drag and drop interface instead of coding the different states a robot can be in and the goal (end state)	 FlexBE Editor - Team VIGIR Edition FlexBE Editor Onboard Status: offline Behavior Dashboard Statemachine Editor Runtime Control Configuration Add State Add Behavior Add State Machine Data Flow Graph Check Behavior Save Behavior Undo Redo Reset Write Comment Toggle Comments TEAM VIGIR Turn Valve (root) Preparation Statemachine 6 states Decide_Walking OperatorDecisionState Walk to Template Walk to Template WalktoTemplateSM Prepare_Manipulation Statemachine 6 states Manipulate_Valve Statemachine 11 states Inform_About_Stand LogState Locomotion Config Locomotion Config LocomotionConfigSM Decide_Step_Back OperatorDecisionState Step_Back Statemachine 2 states finished skip walk failed aborted done stand step_back Apply Close Delete Place the valve template. Prepares the robot for walking to the next task. Decide_Walking OperatorDecisionState Implements a state where the operator has to manually choose an outcome. Autonomy Level of all outcomes should be set to Full, because this state is not able to choose an outcome on its own. Only exception is the suggested outcome, which will be returned immediately by default. This state can be used to create alternative execution paths by setting the suggestion to High autonomy instead of Full. Parameters outcomes: ['walk', 'skip'] hint: "Walk to the valve?" suggestion: 'walk' Required Autonomy Levels walk: High skip: Full Image Ref

NON-ROS BASED SOFTWARE				
Unity	Easy-to-use game engine	C#/Javascript/ Visual scripting	Simplification: Everything in the scene including scripts and objects is stored in a hierarchy. Can be used in conjunction with ROS to place the actual ROS robot into the scene along with URDF files etc Just like RViz/Gazebo except it is in Unity, so it can use Unity features, but this requires a lot of tinkering because ROS support is a plugin, not built directly into unity Mapping: The scene is itself a virtual environment that can be defined Physics: Unity has a physics engine that can affect anything in the scene	 Hierarchy All SampleScene* Main Camera Directional Light Parent Child 1 Child 2 Child 3
Myrobotlab	Open-source framework for robotics and creative machine control	Java (only works with Arduino microcontrollers)	Control: Controls a physical robot and can communicate with other robots. Includes support for speech recognition, text to speech, motor control, servo control, GUI controllers, and microcontroller communications	 myrobotlab - gui system help Welcome gui python runtime services category filters all actuators audio controller intelligence microcontroller network programming robots sensors simulator speech recognition speech synthesis timers vision possible services ACEduinoMotorShield latest AWTRobot latest Adafruit16CServoDriver latest AdafruitMotorShield latest Arduino latest AudioCap AudioFile ChessGa Clock Cortex Cron GUIService latest GoogleSTT latest Graphics latest HTTPClient latest Houston latest IPCamera latest InMoov latest InverseKinematics latest JFugue latest running services gui python runtime Input new service name arduino OK Cancel Image Ref

PACKAGES				
MoveIt!	ROS package used for manipulation. Used in conjunction with RViz. Controls move_group ROS nodes inside MoveGroup Interface separated into MoveIt! Setup Assistant and MoveIt! Commander	Graphical User Interface	Physics: Used to control move_group nodes and manage collisions, joints, and robot poses	 MoveIt Setup Assistant Start Self-Collisions Virtual Joints Planning Groups Robot Poses End Effectors Passive Joints Author Information Configuration Files Robot Poses Create poses for the robot. Poses are defined as sets of joint values for particular planning groups. This is useful for things like *folded arms*. Pose Name: R1Up Planning Group: robot1 robot1_shoulder_pan_joint 0.4143 robot1_shoulder_lift_joint -1.5700 robot1_elbow_joint 0.0000 robot1_wrist_1_joint -1.5700 robot1_wrist_2_joint 0.0000 robot1_wrist_3_joint 0.0000 Image Ref
Open-RMF	Open Robotics Middleware Framework: Open-source modular software system that enables robotic system interoperability (ie, collaboration between robots). These are ROS libraries	Python/C++ (ROS)	Control: Allows multiple ROS systems to communicate and work with each other	 Integration Layer for API normalization Application-specific hardware, firmware, or software Task input adapters (UI portals, etc.) rmf_task_msgs/Delivery rmf_task_msgs/Station alarm system adapters (fire, etc.) std_msgs/Bool rmf_core fleet adapters (tightly integrated with rmf_core) rmf_dispenser_msgs/DispenserRequest rmf_dispenser_msgs/DispenserState rmf_dispenser_msgs/DispenserResult rmf_lift_msgs/LiftRequest rmf_lift_msgs/LiftState rmf_door_msgs/DoorRequest rmf_door_msgs/DoorState rmf_fleet_msgs/ModeRequest rmf_fleet_msgs/DestinationRequest rmf_fleet_msgs/PathRequest rmf_fleet_msgs/FleetState dispenser workcell adapters lift adapters (aka "elevators") door adapters mobile fleet drivers (robots, carts, etc.) dispenser workcells (fixed automation equipment, etc.) lifts (aka "elevators") doors robot fleet managers (AMRs, AGVs, etc.) carts and other things that move and can be tracked / predicted Image Ref

FILETYPES				
URDF	Unified Robot Description: displayed in a tree format (acyclic, directed graph) Describes all elements of a robot and can be placed into Gazebo or Unity to show what the robot actually looks like	XML	Physics: Language of joints and links (joints connect links which give structure) These joints and links can then be referred to in code to perform actions, such as moving an arm	 link A joint A joint B link B link C joint C link D Image Ref

XACRO	XML Macros: must be converted to URDF for usage in ROS	XML	Simplification:
			Simplifies URDF and gets around its limitations with the programmatic generation, templates, parameters, and imports
				
				Consider the following Xacro XML snippet:
				`<xacro:macro name="pr2_arm" params="suffix parent reflect">`
				`  <pr2_upperarm suffix="${suffix}" reflect="${reflect}" parent="${parent}" />`
				`  <pr2_forearm suffix="${suffix}" reflect="${reflect}" parent="elbow_flex_${suffix}" />`
				`</xacro:macro>`
				
				`<xacro:pr2_arm suffix="left" reflect="1" parent="torso" />`
				`<xacro:pr2_arm suffix="right" reflect="-1" parent="torso" />`
				This snippet expands to:
				`<pr2_upperarm suffix="left" reflect="1" parent="torso" />`
				`<pr2_forearm suffix="left" reflect="1" parent="elbow_flex_left" />`
				`<pr2_upperarm suffix="right" reflect="-1" parent="torso" />`
				`<pr2_forearm suffix="right" reflect="-1" parent="elbow_flex_right" />`
				Image Ref