ROS Workspace

Lege einen neuen Workspace an und erzeuge in diesem ein neues Package an. Bei der Namensgebung bitte auf Eindeutigkeit achten (Nachnamen im Name). Hierfür bieten sich die Tutorials 1.1.2 bis 1.1.4 auf http://wiki.ros.org/ROS/Tutorials an.

ROS IDE

Auswahl und Installation der präferierten IDE mit ROS Integration. Es existieren mehrere IDEs mit ROS Support, eine vollständige Liste ist unter http://wiki.ros.org/IDEs zu finden.

Eine einfach zu installierende Option ist Visual Studio Code. Dieses bietet die Extension *Visual Studio Code Extension for ROS* an (https://marketplace.visualstudio.com/items?itemName=ms-iot.vscode-ros).

Aufgabe

Diese Aufgabe vertieft die Themengebiete um das ROS Build System anhand eines Beispiels. Dabei werden mehrere ROS Nodes erzeugt die miteinander interagieren und eine gemeinsame Funktionalität anbieten. Für die Kommunikation werden verschiedene Patterns eingesetzt und der Austausch von Nachrichten erfolgt mittels eigens definierten Nachrichtentypen. Die einzelnen Nodes werden als Publisher, Subscriber und Service implementiert und können über Parameter und Launch-Files gestartet werden. Die Implementierung erfolgt mittels Python (alternativ in C++).

Funktionalität

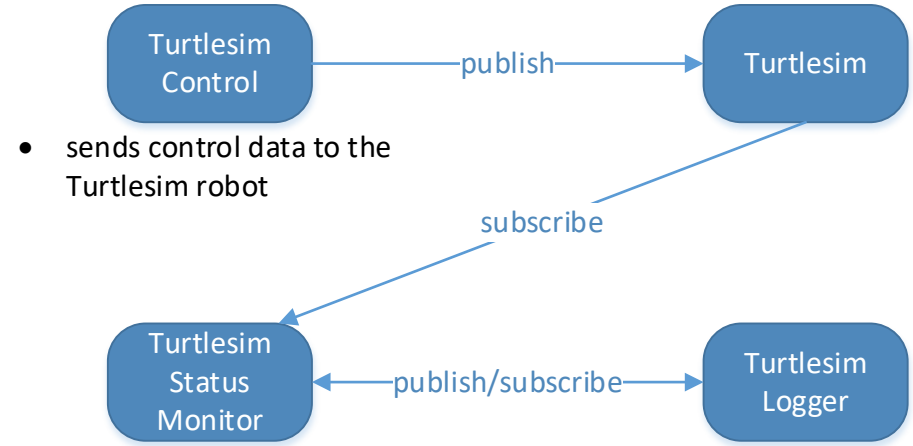
1. Implementieren Sie einen Knoten, der es erlaubt den Turtlesim-Roboter zu steuern. Dies kann über die Tastatur, aber auch über einen Game-Controller oder mittels Daten aus einer Konfigurationsdatei erfolgen (**Publish**).
2. Implementieren Sie einen Knoten der den aktuellen Zustand des Turtlesim-Roboters sammelt (**Subscribe**) und aggregiert.
3. Erstellen Sie einen eigenen (verschachtelten) Nachrichtentyp, der den Gesamtzustand des Systems kapselt und über einen Kanal zur Verfügung gestellt werden kann. (**Custom Messages**)
4. Implementieren Sie einen Knoten, der die Protokollierung des Gesamtzustands erlaubt. Das Datenformat zur Speicherung kann frei gewählt werden (CSV, JSON, Database…). (**Subscribe**)
5. Implementieren Sie einen Service, der es erlaubt die Frequenz der aufzuzeichnenden Daten (Gesamtzustand) festzulegen (**Service Server**). Implementieren Sie einen entsprechenden Client, der diese Frequenz setzen kann. (**Service Client**)
6. Implementieren Sie einen Service, der das Ein- und Abschalten der Steuerung ermöglicht.

Alle Knoten können, müssen aber nicht, im selben Package implementiert werden. Dies erleichtert die Verwaltung von Abhängigkeiten.

Schreiben Sie für die oben spezifizierten Knoten jeweils ein Launch-File, über das die Knoten gestartet werden können.

Verwenden Sie Parameter, wo diese sinnvoll sind. Die Parameter sollen dabei auch über die Launch-Files als Argument angegeben werden können. Mögliche Parameter sind: Frequenz der Updates, Speichern aktivieren, Name des Knoten, Konsolenoutputs aktivieren…

Abbildung 1 zeigt schematisch den Aufbau und die Kommunikation der einzelnen Knoten. Einige Knoten fungieren dabei zusätzlich als Service Server. Ein zusätzlicher Knoten wird als Service Client fungieren und entsprechend die angebotenen Services nutzen.



- sends control data to the Turtlesim robot
- retrieves status from Turtlesim (multiple topics)
- aggregates data into a custom message
- publishes aggregated data
- receives and stores data

Abbildung 1 - Schematischer Aufbau und Zusammenhang der Knoten

Zusatz

Versuchen Sie eine zusätzliche Abhängigkeit von Python über `package.xml` einzubinden und mittels `rosdep` zu installieren. Beispielsweise kann der Protokollierungsknoten live die empfangenen Daten mittels `numpy` und `matplotlib` visualisieren. Alternativ kann der vom Roboter gefahrene Pfad, mittels `pygame` in einem eigenen Fenster visualisiert werden.