**Roboterlabor – Projekt ROS**

**Titel:** Tic Tac Toe Roboter

**Grundidee:** Ein Roboterarm soll Tic Tac Toe spielen. Dafür werden das Spielfeld und Spielsteine in einer Umwelt in Gazebo aufgebaut. Der Roboterarm soll in einer ersten Version des Spieles von Benutzer über das Terminal bedient werden, um ihm zu sagen, wo er die Spielsteine platzieren soll.   
In einer zweiten Version könnte man sich vorstellen, dass der Roboter jeden zweiten Zug autonom durchführt. Der Benutzer wäre in diesem Fall sein Spielgegner.

**Roboter-Art:**

Als Roboterarm wird ein Modell rausgesucht, was z.B. über ros-industrial erhältlich ist.

- zum Beispiel: UR5 Roboter: <https://github.com/ros-industrial/universal_robot>

**Umsetzung:**

Folgende grobe Punkte / Aufgaben wären zu realisieren:

\* Beginn des Spieles in einer Standard Initialsituation: Spielsteine und Roboterarm auf Startposition

\* Spieler 1 (S1) beginnt zu spielen, indem er eine Zahl in einem offenen Terminal eingibt, die der Feldnummer entspricht, wo er sein Spielstein platzieren will.

\* Im Programm sind Koordinaten für jedes Spielfeld hinterlegt. Die passenden Zielkoordinaten werden entsprechend zur angegebenen Nummer ausgesucht. Anhand von ROS MoveIt wird die Trajektorie für den Roboterarm berechnet.  
 Hierbei muss der Roboterarm:

> von der Warteposition zum Spielstein gehen

> diesen greifen

> sich anschließend zum angegebenen Feld bewegen

> den Spielstein ablegen

> in Warteposition zurückkehren

\* Spieler 2 (S2) ist jetzt dran. Eine entsprechende Meldung wird im Terminal angezeigt, damit der Spielerwechsel bekannt gegeben wird. S2 gibt auch über eine Zahl an, wo sein Spielstein (der anderen Farbe) platziert werden soll.

\* S1 ist wieder dran, usw...

\* Wenn erkannt wird, dass jemand gewonnen hat, soll eine entsprechende Meldung im Terminal erscheinen, die den Sieger angibt.

---- Optional, je nach Zeit und Probleme, die eintreten oder nicht :p ----

\* Es wird dann angeboten eine weitere Partie zu starten.

\* Wenn ja ausgewählt wird, werden wieder alle platzierten Spielsteine vom Roboter vom Spielfeld entfernt und auf die Startposition gestellt.

\* Die neue Partie kann beginnen.

---- Weitere Erweiterung ----

\* Anstatt zwei Spieler zu haben, spielt ein Spieler gegen den Roboter. Dieser entscheidet also selbst, wo er seinen Spielstein platziert.

-----------------------------------

1. World in Gazebo

- Spielfeld definieren: Raster zeichnen

- Spielsteine in Anfangszustand platzieren

1. Roboterauswahl

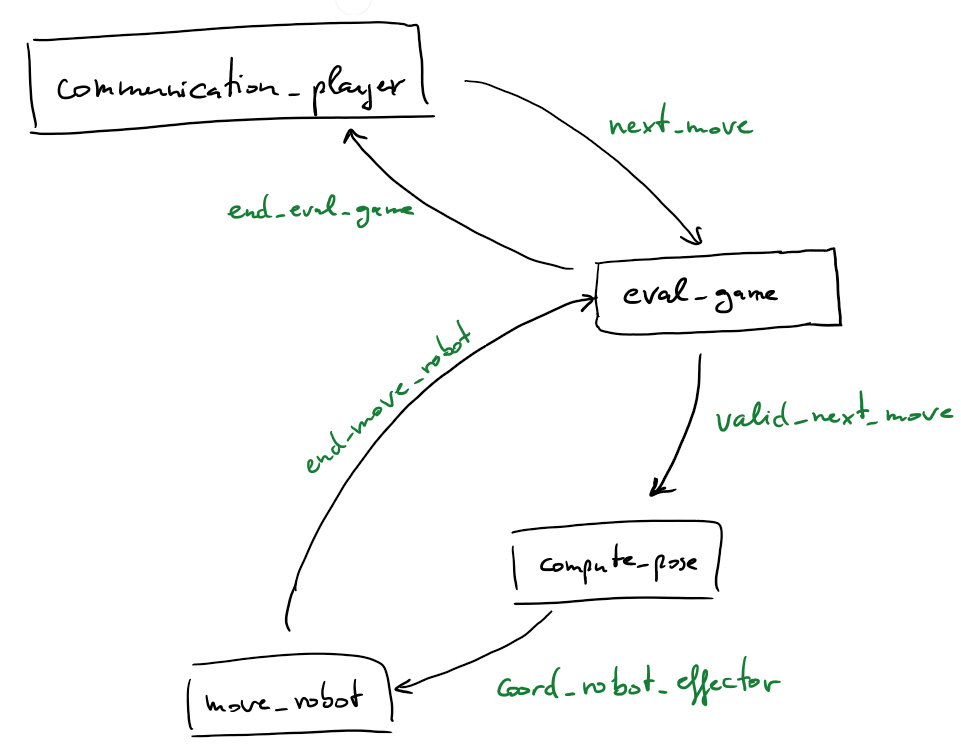
- Roboter: Baxter Research Robot

- Roboter Auswahl + Positionierung

1. Definition der Schnittstellen / Programmaufbau

- ROS Nodes und Topics Definieren evtl. Service festlegen um bidirektionale Kommunikation aufzubauen. Beispiel: Ablage des gegnerischen Spielsteins

Allgemeine Übersicht:



**3.1. Node: communication\_player**

- Start des Programms

* Spieler 1 beginnt
* Wo soll Spielstein gesetzt werden?
* Weitergabe der Spielernummer und Spiefeldnummer durch “next\_move”

- durch Callback-Funktion gestartet, wenn Subscriber auf Topic “end\_eval\_game” eine neue Message erhält

- Kommunikation mit dem Spieler:

* Gibt Info wer dran ist: Spieler 1 oder 2
* Fragt wo der nächste Spielstein platziert werden soll
* Gibt Info, ob das Spiel von jemandem gewonnen wurde oder ob Gleichstand ist  
  Information kommt aus dem Topic “end\_val\_game”, dem subscribed wird

- Speichert die vom Spieler eingegebene Nummer

- Publisher: sendet gewählte Spielfeldnummer und Spielernummer

\* Topic: next\_move

\*Nachrichtentyp: std\_msgs::bool (0 = Spieler 1, 1 = Spieler 2)

Std\_msgs::Char (Spielfeldnummer)

- Beschreibung Topic:

\* Name: next\_move

\* Nachrichtentyp: std\_msgs::Int8MultiArray

\* Inhalt: [Spielfeldnummer (1-9), Spielernummer (1 oder 2)]

- Beschreibung Topic:

\* Name : end\_eval\_game

\* Nachrichtentyp: std\_msgs::String

\* Inhalt: Schlüsselwort, was den aktuellen Spielstand beschreibt

Bsp.: > Ungültig (für ungültige Spielfeldnummer-Angabe durch Spieler)

> Weiter (das Spiel wird ganz normal fortgeführt, weil keiner gewonnen hat, bzw. kein Unentschieden entsteht)

> Gewonnen1 (das Spiel ist beendet: Spieler 1 hat gewonnen)

> Gewonnen2 (das Spiel ist beendet: Spieler 2 hat gewonnen)

> Unentschieden (das Spiel ist beendet: es steht unentschieden)

**3.2. Node: eval\_game**

- durch Callback-Funktion gestartet, wenn:

* Subscriber von Topic “next\_move” (siehe Beschreibung oben)
* Subscriber von Topic “end\_move\_robot”

- enthält Tabelle mit aktueller Platzierung der Spielsteine und Zuordnung zu den Spielern

(0: kein Spielstein, 1: Spieler 1, 2: Spieler 2)

- wenn “next\_move”:

* Überprüft, ob nicht bereits ein Spielstein auf dem angegebenen Feld steht
* Publisher: sendet “true” oder “false”

Topic: valid\_next\_move

- wenn “end\_move\_robot”:

* Überprüft, ob ein Spieler gewonnen hat oder es unentschieden steht
* Publisher: sendet Rückmeldung Richtung Kommunikationsnode

Topic: end\_eval\_game

- Beschreibung Topic:

\* Name: end\_move\_robot

\* Nachrichtentyp: std\_msgs::Bool

\* Inhalt: wurde die Bewegung vom Spielstein erfolgreich durchgeführt

> True: alles ok

> False: Problem

- Beschreibung Topic:

\* Name: valid\_next\_move

\* Nachrichtentyp: std\_msgs::Int8

\* Inhalt: Spielfeldnummer des nächsten Spielsteins

> Nummer zwischen 1 und 9

**3.3. Node: compute\_pose**

- durch Callback-Funktion gestartet, wenn der Subscriber auf dem Topic “valid\_next\_move” eine neue Nachricht bekommt

- enthält eine Tabelle mit der Korrespondenz zwischen den Spielfeldnummern und den Koordinaten, die für die Bewegung des Roboters benötigt werden.

- berechnet eventuell die neue Rotation des Effektors damit der nächste Spielstein ohne Kollision platziert werden kann

- Publisher: sendet die x-, y-, z-Koordinaten für den Roboter und eventuell die Rotationswinkel falls benötigt...

Topic: coord\_robot\_effector

- Beschreibung Topic:

\* Name: coord\_robot\_effector

\* Nachrichtentyp: nav\_msgs::Odometry

\* Inhalt: Koordinaten des Effektors für die Spielstein-Platzierung (aus Tabelle anhand der Spielfeldnummer rausgelesen)

> X, Y, Z Koordinaten

**3.4. Node: move\_robot**

- durch Callback-Funktion gestartet, wenn Subscriber auf Topic “coord\_robot\_effector” eine neue Message erhält

- startet die Bewegung des Roboters, um den neuen Spielstein zu greifen und ihn an der gefragten Stelle zu platzieren

- Roboter wird am Ende auf Warteposition zurückgefahren

- Publisher: sendet Info, dass Bewegung fertig ist

Topic: end\_move\_robot

Eventuell noch hinzuzufügen: letzter Stand: Re-Initialisierung

- Hier würde nach dem letzten Spielzug (wenn die Partie als gewonnen oder Gleichstand gewertet wurde) der Ausgangszustand wiederhergestellt werden.

- Hier werden die Spielsteine wieder auf ihre Anfangspositionen platziert und das Spielfeld frei geräumt.

**Autonomer-Roboter:**

* Hier wird ein Tic-Tac-Toe Bot erstellt. Ich denke wir können diesen Code ziemlich einfach für unseren Roboter verwenden.

<https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/>