Inhaltsverzeichnis

1 Feature List

1.1 Fernsteuerung

1.1.1 Beschreibung

Ein Roboter mit EEROS kann über ROS ferngesteuert werden. Dazu sollen standard-ROS-Knoten, wie etwa 'joy'-Packet[1] verwendet werden können.

1.1.2 Teilziele

1. Steuerungsbefehle mit einem einfachen ROS-Knoten wie der *turtle_teleop_key*.
2. Der Steuerungsbefehl wir über die HAL eingelesen. Die Steuerungsbefehle können als reguläre Inputs im EEROS gelesen werden.
3. Steuerungsbefehle mit einem generischen Tastatur-Knoten damit eine Steuerung über eine Tastatur möglich ist.
4. Steuerungsbefehle mit einem XBox-Controller.

1.2 Logging (Protokollierung)

1.2.1 Beschreibung

Mit *Logging* sind Ausgaben gemeint, die den aktuellen Status der EEROS-Applikation wiedergeben. Sie können auch für Fehlermeldungen und Debug-Informationen verwendet werde.

Das EEROS-Framework hat bereits eine Logger-Funktionalität. In der aktuellen EEROS-Version kann der *Loggers* mit folgenden Zeilen benutzt werden[2]:

```
1 StreamLogWriter w(std::cout); Logger log(); log.set(w);
2
3 log.info() << "Logger Test";
4 int a = 298;
5 logg.warn() << "a = " << a;
6 log.error() << "first line" << endl << "second line";
```

In EEROS erfolgt die Ausgabe des *Loggers* über die Konsole (*StreamLogWriter*) oder in eine Datei (*SysLogWriter*). Wenn die Ausgabe auf einem Anderen PC erfolgen soll, z.B. bei einem ferngesteuertem Roboter, kann eine SSH-Verbindung hergestellt werden.

EEROS bietet auch eine Möglichkeit um Informationen im *Control System* zu loggen[3]. Dabei ist das Problem, dass das *Control System* normalerweise sehr oft (1000 Mal in der Sekunde) ausgeführt wird und den Bildschirm mit Informationen überfluten würde. Es existiert aber bereits eine Lösung mit *Periodic Functions*, damit die Daten mit einer viel kleineren Frequenz ausgegeben werden. Die Lösung mit den *Periodic Functions* ist aber nicht intuitiv und wird oft, besonders in der Debugging-Phase nicht genutzt und umgangen.

ROS hat mit der *ROS console*[4] eine ausgereifte Logging-Funktion integriert. Mit der *Throttle-Funktion* (`ROS_DEBUG_THROTTLE(period, ...)` bietet ROS eine sehr bequeme Möglichkeit, um eine Information nur einmal in einem bestimmten Zeitraum auszugeben. Weitere Funktionen sind noch:

- ROS_DEBUG_COND(cond, ...)
- ROS_DEBUG_ONCE(...)

[1] http://wiki.ros.org/joy
[2] http://wiki.eeros.org/tools/logger/start?s[]=log
[3] http://wiki.eeros.org/tools/logger_cs/start
[4] http://wiki.ros.org/rosconsole

- ROS_DEBUG_DELAYED_THROTTLE(period, ...)
- ROS_DEBUG_FILTER(filter, ...)

Wie auch EEROS hat ROS verschiedene *verbosity levels* um Debug-Informationen von normalen Informationen, Warnungen und Fehlern zu unterscheiden.

Ein weiterer Vorteil bei ROS ist, dass die Ausgaben irgendwo im ROS-Netzwerk, also auch auf einem anderen PC, gelesen und in eine Datei gespeichert werden können. Zusätzlich existieren schon ausgereifte Programme, welche die Ausgaben live filtern, farblich hervorheben und ausgeben können.

1.2.2 Teilziele

Dieses Feature hat einen sehr grossen Funktionsumfang und wird deshalb in mehrere Teilziele unterteilt.

1. Bestehender EEROS-Logger umlenken in den ROS-Logger
2. Bestehender EEROS-Logger umlenken in den ROS-Logger und die *verbosity levels* beibehalten
3. Bestehender EEROS-Logger erweitern um die *Throttle*-Funktionalität
4. Bestehender EEROS-Logger erweitern um die *Conditional*-Funktionalität
5. Bestehender EEROS-Logger erweitern um die *Once*-Funktionalität
6. Bestehender EEROS-Logger erweitern um die *Filter*-Funktionalität
7. Bestehender EEROS-Logger erweitern um die *Delayed-Throttle*-Funktionalität

1.3 Anzeige von Prozessvariablen

1.3.1 Beschreibung

Prozessvariablen sind, im Gegensatz zu den Log-Ausgaben, einzelne Zahlen oder Datenpunkte wie beispielsweise die Position eines Encoders. Die Variablen können in einem GUI angezeigt werden. Im einfachsten Fall zeigt das GUI die Variablen in einer einfachen Konsole an. In vielen Fällen ist es aber auch von Vorteil, wenn eine oder mehrere Variablen in einem Graphen visualisiert werden können.

1.3.2 Zu erwartende Probleme

Bei Prozessvariablen gilt es zu beachten, dass sehr schnell eine grosse Menge von Daten anfallen können. Dabei ist nicht nur die Bandbreite ein Problem, sondern auch die Latenz. Im konkreten Fall bedeutet dies, dass in einem *Control System* bei jedem Durchlauf innerhalb von sehr kurzer Zeit (typischerweise 1 s) neue Daten produziert werden. Wenn das ROS-Netzwerk nicht innerhalb von einer Millisekunde die Daten wegschicken kann, kann es sein, dass Daten verloren gehen.

Eine Lösung für die zu geringe Latenz des ROS-Netzwerk dazu wäre ein Buffer, der den hochfrequenten Datenstrom abfängt und von in längeren Zeitabständen (etwa 0.1 s bis 1 s) Datenpakete mit den Daten schickt.

Wenn aber die Bandbreite zu hoch ist, das heist, wenn mehr Daten durch das ROS-Netzwerk geschickt werden, als das Netzwerk übertragen kann, dann reicht ein einfacher Buffer nicht mehr aus. Folgende Techniken könnten das Problem lösen:

- **Throttle:** Funktioniert wie der Logger mit *Throttle*-Funktionalität. Die meisten Daten werden verworfen und nur jeder x-te Wert wird geschickt
- **Zeitbegrenzter Buffer:** Ein Buffer speichert über eine begrenzte Zeit die Daten und schickt sie dann mit reduzierter Bandbreite über das Netzwerk..
- **Filter:** Es werden nur Daten gesendet, die eine bestimmte Bedingung erfüllen. Z.B. wenn deren Wert grösser als 10 ist.

- **Statistik:** Für eine gewisse Anzahl von Datenpunkte werden statistische Werte wie z.B. Mittelwert, Minimum und Maximum berechnet. Dem Netzwerk werden nur die berechneten Werte gesendet.

1.3.3 Teilziele

1. Anzeige von Daten in einer Konsole (über das ROS-Netzwerk)
2. Anzeige von Daten in einem Diagramm (bestehende ROS-Tools)
3. Anzeige in Gazebo
4. Throttle
5. Zeitbegrenzter Buffer
6. Filter
7. Statistik

1.4 Variablen in EEROS per Fernsteuerung manipulieren

1.4.1 Beschreibung

Ein EEROS-Roboter hat diverse Konstanten, wie etwa Umrechnungsfaktoren und Offsets, gespeichert, die das Verhalten des Roboters beeinflussen. Für die Fehlersuche ist es manchmal von Vorteil, wenn Ausgänge und Konstanten manuell gesetzt werden können.

1.4.2 Teilziele

1. Ausgänge ferngesteuert setzen.
2. Konstanten ferngesteuert setzen.

1.5 Bewertung der verschiedenen Features und Teilziele

Feature	**Teilziel**	**Nutzen**	**Aufwand**	**Punkte**
Fernsteuerung	1. Einfacher ROS-Knoten	6	3	2.0
	2. HAL	8	7	1.1
	3. Generischer Tastaturknoten	7	5	1.4
	4. XBox-Controller	7	5	1.4
Logging	1. EEROS-Logger umlenken	8	4	2.0
	2. *Verbosity levels* beibehalten	7	5	1.4
	3. *Throttle*-Funktionalität	8	6	1.3
	4. *Conditional*-Funktionalität	4	3	1.3
	5. *Once*-Funktionalität	4	3	1.3
	6. *Filter*-Funktionalität	3	3	1.0
	7. *Delayed-Throttle*-Funkt.	3	3	1.0
Anzeige von Prozessvar.	1. Konsolenausgabe	8	5	1.6
	2. Diagramm	8	6	1.3
	3. Gazebo	4	9	0.4
	4. Throttle-Funktion	8	6	1.3
	5. Zeitbegrenzter Buffer	6	6	1.0
	6. Filter	6	6	1.0
	7. Statistik	7	7	1.0
Manipulieren von Prozessvar.	1. Ausgänge setzen	8	4	2.0
	2. Konstanten setzen	6	4	1.5

2 Einbindung in EEROS

2.1 CMAKE

2.1.1 Erkennen ob ROS installiert ist

Diejenigen Teile von EEROS welche ROS-Bibliotheken verwenden, können nur kompiliert werden, wenn ROS auch auf dem System installiert ist. CMAKE nutzt dafür den "*find_package()*-Befehl. *roscpp* ist nur ein einzelnes *Package* und nicht das ganze ROS-Framework. Wenn aber *roscpp* gefunden wird, kann davon ausgegangen werden, dass auch das restliche Framework installiert wurde.

```
1 message(STATUS "looking for package 'ROS'")
2 find_package(roscpp QUIET)
3 if (roscpp_FOUND)
4   message( STATUS "-> ROS found")
5   message( STATUS "roscpp_DIR:  " ${roscpp_DIR})
6 endif()
```

2.1.2 Mögliche Probleme

Bevor CMAKE das *Package* finden kann, muss der *Setup-Script* von ROS mit folgendem Befehl ausgeführt werden:
 # source /opt/ros/kinetic/setup.bash

Anhang