**Oberstufenlabor Mechatronik I – Laborversuch mit dem youBot**

**Dokumentation**

**Allgemeine Informationen**

Alle für den Versuch benötigten Dateien befinden sich im GitLab unter: <http://130.75.144.241/koehler/youbot_mechatroniklabor>

Das Programm für den Versuch ist in drei Packages aufgeteilt:

1. *oberstufenlabor\_mechatronik\_camera\_calibration*: Enthält das Programm für die Kalibrierung der Kamera
2. *oberstufenlabor\_mechatronik\_vision*: Enthält das Programm zur Bilderkennung / Kreisdetektion
3. *oberstufenlabor\_mechatronik\_robot*: Enthält den Server für die Robotersteuerung und das Sammeln der Bälle

Alle Aufgabenteile und dazugehörige Hinweise / Links sind zusätzlich im Quelltext der jeweiligen Programme enthalten.

Die Dateien ohne „\_studis“ im Namen enthalten die Musterlösung.

Allgemeines Vorgehen beim Kompilieren:

Mit dem Befehl „*cd ~/catkin\_ws*“ in den Catkin-Workspace wechseln und den Befehl „*catkin\_make*“ ausführen.

Editieren der Dateien entweder mit dem Standard-Texteditor oder alternativ mit dem Qt Creator.

Der Zugriff auf den youBot 3 erfolgt über den Befehl "*ssh -X* [*imesadmin@192.168.1.230*](mailto:imesadmin@192.168.1.230)". Das Passwort ist "*youbot*". Die Dateien für das Labor befinden sich auf dem youBot im Ordner "*ros\_indigo/catkin\_ws/src/youbot\_mechatroniklabor*".

Der Treiber für den Greifer, Arm, Base und Laserscanner des youBots muss zuerst gestartet werden. Dazu erst den Befehl "*sudo udevadm trigger*" (Passwort: youbot) und dann den Befehl "*roslaunch oberstufenlabor\_mechatronik\_robot base\_system.launch*" auf dem youBot ausführen.

Dateien können über SSH mit dem Befehl "scp Datei [imesadmin@192.168.1.230:/home/imesadmin/](mailto:imesadmin@192.168.1.230:/home/imesadmin/)..." auf den youBot kopiert werden.

**Aufgabe 1 – Kamerakalibrierung**

Zur Vorbereitung auf die nachfolgenden Aufgaben aus dem Gebiet der Bildverarbeitung ist die verwendete Kamera des youBots zu kalibrieren. Das ROS-Package

„*oberstufenlabor\_mechatronik\_camera\_calibration*“ bietet die Möglichkeit

zur Kalibrierung einer Kamera mit Hilfe eines Schachbrettmusters. Das hier verwendete Schachbrettmuster besteht aus insgesamt 63 Quadraten mit einer

Kantenlänge von 30 mm, die innenliegend 8 x 6 Kanten beinhalten. Die ROS-Node zur Kamerakalibrierung wird über folgenden Befehl gestartet:

„*roslaunch oberstufenlabor\_mechatronik\_camera\_calibration exercise01\_camera\_calibration.launch*“

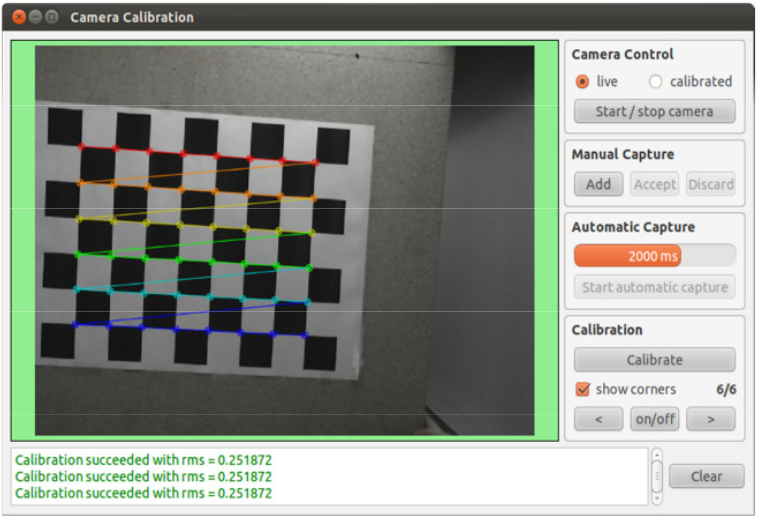


Abbildung 1: Benutzeroberfläche

Es erscheint die grafische Benutzeroberfläche aus Abbildung 1. Das Kamerabild wird nach einem Klick auf die Schaltfläche „*Start/stop camera*“ angezeigt.

Nach Betätigung der Schaltfläche „*Start automatic capture*“ werden im Abstand von drei Sekunden automatisch Bilder aufgenommen. Die Kalibrierung sollte wie in Abbildung 2 dargestellt durch eine halbkugelförmige Bewegung des Schachbrettmusters um die Kamera erfolgen.

Durch einen Klick auf „*Stop automatic capture*“ lässt sich die automatische Bildaufnahme beenden. Nach dem Deaktivieren des Kamerabildes über „*Start / stop camera*“ können die aufgenommenen Bilder vor dem Start der Kalibrierung über die Schaltflächen „*<*“ und „*>*“ durchgesehen werden und bei Bedarf mit Hilfe der Schaltfläche „*on / off*“ (de)aktiviert werden.

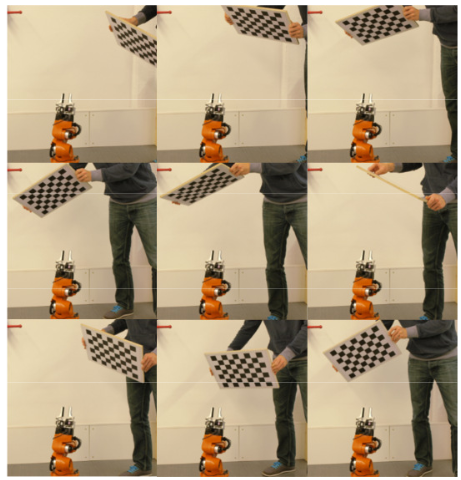


Abbildung 2: halbkugelförmige Kalibrierbewegung

Die Kalibrierung erfolgt im Anschluss über einen Klick auf „*Calibrate*“. Eine Statusmeldung gibt dabei Auskunft darüber, ob die Kalibrierung erfolgreich war.

Die Kalibrierungsparameter der Kamera D, K, R und P können über die Auswahl des Menüeintrags „*Export*“ → „*ROS Parameterset*“ in einer .yaml-Datei gespeichert werden. Zum Laden der Kalibrierungsdatei, sollte die Template-Datei „*head\_camera.yaml*“ in

„*oberstufenlabor\_mechatronik\_camera\_calibration/resources/*“ mit den Parametern gefüllt werden. Kopieren Sie die Datei im Ordner „*~/.ros/camera\_info/*“. Im Dateimanager ist dieser Ordner standardmäßig versteckt. Es muss deshalb die Option „*Versteckte Dateien anzeigen*“ aktiviert werden, falls die Datei über den Dateimanager kopiert wird.

**Aufgabe 2 - Bildverarbeitung**

Die Aufgabe 2 beschäftigt sich mit der Bildverarbeitung. Die Studenten sollen zum einen Lücken im Programm ergänzen bzw. füllen und zum anderen mit einem Programm die von ihnen erstellten Programmteile testen und die Parameter für die Farbfilterung und die morphologischen Operationen festlegen.

Beim Programmierteil sind folgende Teilaufgaben zu erledigen:

* Umwandlung des Farbraums eines Bildes von BGR in HSV
* Anwendung von Farbfiltern
* Anwendung von morphologischen Operationen
* Konturendetektion / Erkennung eines Kreises
* Berechnung von Radius und Mittelpunkt
* Einzeichnen des gefundenen Kreises in eine Bildmatrix

Der dazugehörige Quellcode befindet sich in der Datei „*/oberstufenlabor\_mechatronik\_vision/src/exercise02\_library\_studis.cpp*“. Das Programm zum Testen kann über den Befehl „*roslaunch oberstufenlabor\_mechatronik\_vision exercise02.launch*“ gestartet werden.

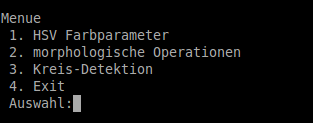


Abbildung 3: Auswahlmenü

Abbildung 3 zeigt das Menü zur Auswahl der einzelnen Teilaufgaben.

**Achtung: Der Arm fährt zu Beginn des Programms in seine Such-Pose und am Ende wieder in die Home-Stellung. Bitte genügend Abstand halten!**

Es folgen die Teilaufgaben, die so 1:1 auch im Quelltext stehen:

1. In der Funktion „*exercise02\_library::filterImage*“:

Führen Sie eine Farbraumkonvertierung der Bildmatrix "*image*", die im BGR-Format vorliegt, in den HSV-Farbraum durch. Deklarieren Sie hierzu eine Bildmatrix zur Aufnahme der HSV-Bilddaten.

Implementieren Sie einen Farbfilter im HSV-Farbraum. Achtung: Die Schwellwerte hierfür befinden sich in den Variablen "*hsv\_min*" und "*hsv\_max*", deren Werte mit Hilfe der bereits integrierten Schieberegler gesetzt werden. Die gefilterte Matrix soll in die Bildmatrix "*colorFilter*" (siehe oben) kopiert werden.

Hinweise:

<http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html>

<http://docs.opencv.org/modules/core/doc/operations_on_arrays.html>

<http://docs.opencv.org/modules/core/doc/basic_structures.html>

Zum Testen kann das Programm wie oben beschrieben über das Launch-File gestartet werden. Die Eingabe der Zahl „1“ startet die Bestimmung der Farbparameter. Wurden alle Parameter über die Schieberegler für alle drei Farben eingestellt, wird das Programm beendet.

1. In der Funktion „*exercise02\_library::morphImage*“*:*

Die Bildmatrix "*image*" wurde in Aufgabenteil A bereits in den HSV-Farbraum konvertiert und anschließend wurde eine Farbfilterung im HSV-Farbraum durchgeführt.

Auf das resultierende Bild sind nun die morphologischen Operationen "Erosion" und "Dilatation" nacheinander anzuwenden. Die Integervariablen "*erosion\_size*" und "*dilation\_size*" enthalten die Kernelgrößen. Sie können später über Schieberegler eingestellt werden.

Die Eingangsmatrix "*image*" soll gleichzeitig als Ausgangsmatrix dienen.

Hinweise:

<http://docs.opencv.org/doc/tutorials/imgproc/table_of_content_imgproc/table_of_content_imgproc.html>

<http://docs.opencv.org/modules/imgproc/doc/filtering.html>

Das Ergebnis kann getestet werden, in dem im Menü des Programms die Zahl „2“ eingegeben wird. Über die Schieberegler lassen sich die Kernelgröße einstellen.

Aufgabenteil C) und D) müssen nacheinander abgearbeitet werden, erst dann kann das Programm getestet werden.

1. In der Funktion „*exercise02\_library::searchForCircles*“:

Die Eingangsmatrix "*image*" soll nun auf Konturen hin untersucht werden, um den Ball bzw. Kreis im Bild zu erkennen.

1. Konvertieren Sie die Matrix "*image*" von RGB zu Graustufen. Durchsuchen Sie anschließend das konvertierte Bild nach Konturen. Gefundene Konturen sollen im Vektor "*contours*" (siehe oben) gespeichert werden.
2. Ermitteln Sie den kleinsten Kreis für die gefundenen Konturen. Die Variable "*i*" gibt hierbei die jeweils aktuelle Kontur im Vektor "*contours*" an. Der Mittelpunkt und Radius des Kreises soll in den von Ihnen zu deklarierenden Variablen "*center*" und "*radius*" gespeichert werden. Entfernen Sie außerdem die Kommentierung ab der gekennzeichneten Zeile.

Hinweise:

<http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html>

1. In der Funktion „*exercise02\_library::drawCircle*“:

In die Eingangsmatrix "*image*" soll der gefundene Kreis eingezeichnet werden. Die Informationen über Mittelpunkt und Radius befinden sich in der Variablen "*circle*". Verwenden Sie eine beliebige Farbe, Liniendicke und -typ für den Kreis.

Hinweise:

<http://docs.opencv.org/modules/core/doc/drawing_functions.html>

Das Ergebnis kann getestet werden, in dem im Menü des Programms die Zahl „3“ eingegeben wird.

**Aufgabe 3 – Bilderkennung**

In Aufgabe 3 geht es um die Implementierung der in Aufgabe 2 erstellten Funktionen. Dabei müssen folgende Teilaufgaben bearbeitet werden:

* Überprüfen ob Bild vorhanden, Farbe vorgegeben und Parameter vorhanden
* Einbindung der in Aufgabe 2 erstellten Funktionen zur Erkennung eines Balls
* Publishen der Daten des gefundenen Balls

Der dazugehörige Quellcode befindet sich in der Datei „*/oberstufenlabor\_mechatronik\_vision/src/exercise03\_studis.cpp*“. Ein Programm zum Testen existiert nicht. Die Funktion muss dann später im Gesamtablauf validiert werden.

In der Funktion „*exercise03::findBalls*“:

Die Funktion "*findBalls()*" zum Finden eines Balls in einer vorgegebenen Farbe soll erstellt werden.

1. Zu Beginn sollen folgende Fälle überprüft werden:
   * + Ist die Bildmatrix C mit Daten gefüllt?
     + Wurde in der Variablen "*desired\_color\_*" die gewünschte Farbe übergeben?
     + Wurden die Filter-Parameter für die gewünschte Farbe sowie die Parameter für die morphologischen Operationen gefunden?

Sollte einer dieser Fälle nicht eintreffen, soll sich die Funktion beenden.

Hinweise:

* + - Ob die Parameter für Farbfilter und morphologische Operationen vorhanden sind, kann mit den Funktionen "*findFilterParameter*" und "*findMorphParameter*" überprüft werden. Diese geben eine boolesche Variable mit dem Ergebnis zurück.
    - Wurde keine Farbe übergeben, dann steht in der Variablen "*desired\_color\_*" der String "*none*".

1. Implementieren Sie die in Aufgabe 2 entwickelten Funktionen. Kopieren Sie dazu  
   den Programmcode in die gekennzeichneten Bereiche der Funktionen "filterImage", "morphImage" und "searchForCircles". Der gefundene Kreis soll am Ende an die Variable "*tracked\_circle\_*" übergeben werden.

Das Einzeichnen des gefundenen Kreises in eine Bildmatrix ist nicht nötig!

Der gefundene Kreis kann über die Funktion "*publishCircle*" gepublished werden.

Hinweise:

[http://docs.opencv.org/modules/core/doc/basic\_structures.html#ma](http://docs.opencv.org/modules/core/doc/basic_structures.html#mat)t

<http://www.cplusplus.com/reference/string/string/>

[http://www.cpp-entwicklung.de/cpplinux/cpp\_main/node4.html#SECTION00440000000000000000](http://www.cpp-entwicklung.de/cpplinux/cpp_main/node4.html)

**Aufgabe 4 – Roboterregelung**

In Aufgabe 4 soll eine Statemachine für den Greif- und Ablegeprozess eines Balls entwickelt werden. Diese wird zum Testen und Ausprobieren zunächst in einer extra Datei eingefügt. Dafür gibt es die Dateien "*oberstufenlabor\_mechatronik\_robot/src/oberstufenlabor\_mechatronik\_robot/exercise04\_gripper\_test\_studis.cpp*" und "*oberstufenlabor\_mechatronik\_robot/include/oberstufenlabor\_mechatronik\_robot/exercise04\_gripper\_test\_studis.h*". In der Header-Datei (.h) können die einzelnen Zustände der Statemachine deklariert werden und in der Cpp-Datei die dazugehörigen Befehle.

In der Funktion „*exercise04\_gripper\_test::gripRoutine*“:

An dieser Stelle soll eine Statemachine für den Greif- und Ablegeprozess eines Balls erstellt werden. Die dafür benötigten States können dafür in der Datei "*exercise04\_gripper\_test.h*" definiert werden.  
  
 Hinweise:

* Der Zugriff auf den Arm und den Gripper erfolgt über die Objekte "*arm\_*" und "*gripper\_*". Für den Arm stehen die Posen "*GRIP*" und "*GRIP\_ABOVE*" zur Verfügung. Auf die Posen für den Cargo Slot kann über das Objekt "*current\_cargo\_slot\_*" mit "*above\_pose*" und "*pose*" zugegriffen werden.
* Für die Überprüfung, ob sich der Arm / Gripper in Bewegung befindet, enthalten beide Objekte jeweils die Funktion "*isBusy()*".
* Der Gripper kann über die Funktion "*setWidth()*" geschlossen werden. Die Variable "*grip\_width\_*" enthält die benötigte Breite. Die Funktion "*open()*" öffnet den Gripper.

Zum Anfahren der benötigten Posen existiert das Package "*luh\_youbot\_gui*". Damit können die Posen vorab getestet werden. Zum Starten muss der Befehl "*rosrun luh\_youbot\_gui luh\_youbot\_gui*" ausgeführt werden. Die Posen müssen über die Schaltfläche "*Load...*" zunächst geladen werden. Das zugehörige File befindet sich im Ordner "*oberstufenlabor\_mechatronik\_robot/poses/poses.yaml*". Mit einem Klick auf "*Replace*" werden die Posen geladen und können mit einem Klick auf den grünen Pfeil ausgeführt werden.

Zum Testen der Statemachine kann das passende Launchfile mit "*roslaunch oberstufenlabor\_mechatronik\_robot exercise04\_gripper\_test.launch*" aufgerufen werden.

Wenn der Greif- und Ablegeprozess problemlos funktioniert, dann können die Statemachine und die dazugehörigen States per Copy & Paste in die Dateien "*oberstufenlabor\_mechatronik\_robot/src/oberstufenlabor\_mechatronik\_robot/exercise04\_collector\_server\_studis.cpp*" und "*oberstufenlabor\_mechatronik\_robot/include/oberstufenlabor\_mechatronik\_robot/exercise04\_collector\_server\_studis.h*" übertragen werden.

**Starten des Gesamtprogramms**

Zum Starten und Testen des gesamten Programms müssen folgende Befehle jeweils in einem eigenen Terminalfenster ausgeführt werden (base\_system.launch sollte schon laufen):

1. roslaunch oberstufenlabor\_mechatronik\_robot vision.launch
2. roslaunch oberstufenlabor\_mechatronik\_robot collector\_server.launch
3. rosrun oberstufenlabor\_mechatronik\_robot exercise04\_collector\_node Color number

Bei der Collector-Server-Node müssen die Argumente "*Color*" und "*number*" angegeben werden. Als Argument "*color*" stehen Red, Green und Yellow zur Auswahl, mit dem Argument "*number*" kann die Anzahl der zu findenden Bälle angegeben werden.

**Wichtig: Die gewünschte Farbe muss mit einem großen Anfangsbuchstaben beginnen!**

**Anhang**

**Manipulator Usage**

Basic Position Commands

Position commands for the arm can be set using the moveToPose() method which has four overloads:

* **for jointspace coordinates:**  
  void moveToPose(luh\_youbot\_kinematics::JointPosition pose, int mode=MotionMode::PLAN, bool pose\_is\_relative=false);
* **for cylindrical coordinates:**  
  void moveToPose(luh\_youbot\_kinematics::CylindricPosition pose, int mode=MotionMode::PLAN, bool pose\_is\_relative=false);
* **for cartesian coordinates:**  
  void moveToPose(luh\_youbot\_kinematics::CartesianPosition pose, int mode=MotionMode::PLAN, std::string frame\_id="arm\_link\_0", bool pose\_is\_relative=false);
* **for predefined poses:**  
  void moveToPose(std::string pose\_name, int movement\_mode=MotionMode::PLAN, int interpolation\_mode=InterpolationMode::JOINTSPACE);

The arguments are:

* **pose:** A pose defined in jointspace, cartesian or cylindrical coordinates.
* **mode:** The motion mode can be
  + MotionMode::DIRECT: Direct position command or
  + MotionMode::INTERPOLATE: interpolated movement
* **pose\_is\_relative:** If this is true the pose is interpreted as relative to the current pose.
* **frame\_id:** The frame ID of a cartesian pose used for tf. The default is „arm\_link\_0“, but e.g. „gripper\_tip“ might also be useful in some situations.
* **interpolation\_mode:** This parameter is only needed for predefined poses and interpolated movements. The interpolation mode can be
  + InterpolationMode::JOINTSPACE,
  + InterpolationMode::CYLINDRIC or
  + InterpolationMode.:CARTESIAN.

**Motion Mode**

Two movement modes are available.

* **Direct Position Command:** In this mode only the goal position is set as position command. No interpolation or trajectory planning is done. This mode only works if the ramp generator of the motor controllers is enabled (the parameter can be set in luh\_youbot\_manipulation). Enabling the ramp generator is discouraged however because it significantly reduces the performance of other motion modes.  
  **Advantages**: fast movement  
  **Disadvantages**: unpredictable trajectory; doesn't work together with other motion modes
* **Interpolated movement:** In this mode the trajectory between start and goal point is interpolated in the specified coordinate system. In case of jointspace coordinates this means that the movement of all joints is synchronized which leads to a smoother trajectory and thereby to a better predictability of the movement. In cartesian coordinates the endeffector will be moved along a straight line. Cylindrical coordinates are somewhere in between but closer to cartesian.  
  **Advantages:** good predictability of trajectory, precise movement possible  
  **Disadvantages:** in case of cartesian or cylindrical coordinates only useful for small movements, no collision avoidance

**Movement Execution**

The moveToPose() method starts the movement action and returns immediately. While the action is executing you have several options.

* You can abort the action at any time:

arm.abortCurrentAction();

* You can check if the action is still being executed:

if(arm.isBusy()) //...

* Or you can wait for the action to finish with a specified timeout. The default timeout is zero which means no timeout and the method will only return if the action finishes. The method returns true if the action has finished before the timeout.
  + no timeout:  
    arm.waitForCurrentAction();
  + 10 seconds timeout:  
    if(!arm.waitForCurrentAction(10)) //...
* You can e.g. wait for five seconds and abort the action if it has not finished.

if(!arm.waitForCurrentAction(5))

arm.abortCurrentAction();

* When the action has finished you can check the status. The method actionSucceeded() returns true if the action has succeeded which means the arm has reached the desired goal position. In case anything went wrong during the execution the method will return false.

if(arm.actionSucceeded()) //...

**Gripper usage**

The basic usage for the gripper is the same as for the arm and base.   
This tutorial only introduces the gripper-specific functions.

Gripper commands

There are two methods to control the gripper:

* void open();

This opens the gripper to the maximum width.

* void setWidth(double value);

This sets the gripper width to the specified value in meters.

With the current gripper design it is not possible to get the actual gripper state. For that reason the gripper action can not actually check if the desired position was reached. Instead, it just waits a defined period and then returns.