Technische Universität Braunschweig

Teamprojekt

Koodinierte Ballsuche mit mobilen Robotern

Martin Mikolas, Markus Reschke, Johannes Starosta

Betreuer: René Iser

8. Dezember 2011

Institut für Robotik und Prozessinformatik

Prof. Dr. F. Wahl

Erklärung

Hiermit erklären wir, dass die vorliegende Arbeit selbständig nur unter Verwendung der aufgeführten Hilfsmittel von uns erstellt wurde.

Braunschweig, den 8. Dezember 2011

Unterschrift

Zusammenfassung

Bei der vorliegenden Ausarbeitung handelt es sich um die Dokumentation unseres Teamprojektes zur ,,Koordinierten Suche mit mobilen Robotern". Wir werden zunächst die Vorarbeiten vorstellen, auf denen wir ausbauten, bevor wir unser System näher vorstellen. Zum Schluss folgt noch eine kritische Bewertung unserer Ergebnisse.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Unsere Aufgabe ist es, eine koordinierte Ballsuche in einen Raum zu implementieren.

Das Ziel der koordinierten Suche besteht darin, dass die im Raum bzw. im Suchterrain befindlichen mobilen Roboter miteinander das gesuchte Ziel finden. Diese Suche setzt voraus, dass sich die Roboter kollisionsfrei im Raum bewegen. Sie dürfen weder mit Objekten aus der Umgebung, noch mit anderen Robotern, die ebenfalls auf der Suche nach dem gegebenen Objekt sind, kollidieren. Zur Umsetzung dieser Aufgabe standen eine Ad-hoc-Lösung und schließlich eine Server-Client-Lösung zu Auswahl, auf die die Entscheidung fiel. Dies ermöglicht eine parallele Entwicklung von Client und Server, was eine einfachere Aufgabenverteilung im Team ermöglicht. Ausserdem kann dann auf den Server eine Visualisierung der Ballsuche erfolgen, indem eine Karte des Raumes, sowie der darin befindlichen Roboter sowie (nach erfolgreicher Suche) des Balls in der GUI des Servers angezeigt wird. Der Client ist dann nur noch für das Abfahren der von Server vorgebenen Postionen, sowie die Lokalisierung im Raum und Ballerkennung in seiner unmittelbaren Umgebung zuständig.

Zum Erreichen des Ziels standen uns mobile Roboter ,,Pioneer-3DX" der Firma ,,adept mobilerobots" zur Verfügung. Dazu konnten wir diverse Libaries des Instituts nutzen. Die wichtigsten schon vorhandenen Komponenten waren aber der SonarPartikelfilter, sowie die Ballerkennung:

- Der SonarPartikelfilter geht auf ein vorheriges Teamprojekt zurück. Mit ihm konnten wir die Position des Roboters im Raum relativ genau bestimmen. Diese diente zum einen als Basis für die Bahnplanung. Zum anderen war dies die Basis zur Bestimmung der Position eines erkannten Balls.
- Die Ballerkennung konnten wir aus der Bachelorarbeit von Tobias Breuer übernehmen. Sie stellte uns eine API zur Verfügung, worüber wir den Ball finden, sowie seine genaue Position bestimmen konnten.

Unsere Softwarearchitektur bestand somit aus folgenden Komponenten:

- Der Server: Er nimmt die Bahnplanung, sowie Kollisionsvermeidung vor und steuert bis zu drei Clients. Gleichzeitig dient er als Visualisierung des aktuellen Status der Ballsuche.
- Der Client: Eine Steuersoftware für die Roboter. Sie ist dafür zuständig, die vom Server vorgebene Bahnplanung umzusetzen. Dafür nutzt er den SonarPartikelfilter, um seine eigene Position im Raum

zu bestimmen. Außerdem nimmt er die eigentliche Ballerkennung vor und bestimmt aus den dabei erhalten Informationen und der durch den Partikelfilter erhaltenen Position die Position des Balles im Raumes.

Im nächsten Abschnitt folgt nun eine genauere Beschreibung unserer Software-Architektur

2 Softwarearchitektur

2.1 Allgemeiner Aufbau

Die Software besteht aus zwei Komponenten:

- Der Server, der die Bahnplanung für die Roboter, sowie die Visualisierung der Ballsuche übernimmt.
- Der Client, der die Ballerkennung, sowie Positionsbestimmung vornimmt. Ausserdem ist er dafür verantwortlich, den Roboter zu steuern. Dazu lässt er den Roboter die durch den Server vorgegebenen Punkte der geplanten Bahn anfahren.

2.2 Server

2.3 Client

Der Client in unserer Client-Server-Architektur ist für das Anfahren der einzelnen Routenpunkte und die Erkennung des Balls zuständig. Dazu haben wir für die Lokalisierung die uns zu Verfügung gestellte Implementierung eines Sonar-Partikelfilters und für die Ballerkennung die Implementierung aus einer Bachelorarbeit von Tobias Breuer verwendet. Wir beschreiben im Folgenden zunächst, wie wir den Partikelfilter und die Ballerkennung genutzt haben. Anschließend wird die Funktionsweise des Clients beschrieben.

2.3.1 Ballerkennung

Die Ballerkennung läuft in einen eigenen Thread im Client, da sie und die Fahrschleife des Roboters sich gegenseitig blockieren würden. Dabei greift sie auf das Kamerabild einer Logitech Webcam Pro 9000 zurück. Nach Initialisierung der Ballerkennung werden folgende Schritte durchgeführt, bis der Ball gefunden oder die Suche abgebrochen wurde:

1. Aktualisieren des aktuellen Kamerabildes
2. Aufruf der Erkennungsmethode der Balldetection `detectBall()`.
3. Abfrage, ob der Ball erkannt wurde mit der Methode `getActualPicProb()`. Liefert diese eine Wahrscheinlichkeit zurück, die größer als 80 % ist, ist die Suche beendet und die Koordinaten des Balls werden wie folgt bestimmt:

(a) Mit den Methode getDistanceToBall() und getAngleToBall() wird die Entfernung (in mm) und der Winkel zum Ball (in Grad von der Mitte des Bildes aus (0°), links positiv, rechts negativ) abgerufen. Daraus wird zusammen mit der aktuellen Position des Roboters die Ballposition bestimmt, dazu wird zuerst der Ausrichtungsvektor des Roboters um den Winkel zum Ball gedreht. Dann wird die Entfernung zum Ball ermittelt, da diese in mm ist, muss die Entfernung in px umgerechnet werden. Dazu dient der Wert pixelPerMm aus der Konfiguration, welcher die Auflösung der Karte in px/mm angibt. Um nun die Ballposition zu erhalten, wird der gedrehte Ausrichtungsvektor (ein Einheitsvektor) mit pixelPerMm · Entfernung zum Ball skalar multipliziert und auf den aktuellen Positionsvektor raufaddiert.

(b) Danach wird die gefundene Position zum Server übertragen.

4. Ist die Wahrscheinlichkeit hingegen kleiner als 80 % wird mit der Ballerkennung fortgefahren.

Wie bereits erwähnt greifen wir bei der Ballerkennung auf die Bachelorarbeit von Tobias Breuer zurück. Dadurch konnten wir einfach die API der von ihm entwickelten Software nutzen, ohne uns selber mit den Details der Ballerkennung beschäftigen zu müssen. Sie verfolgt folgenden Ansatz: Im Kamerabild wird nach roten Pigmenten gesucht. Erreichen diese eine gewisse Anzahl, sucht die Software nach kreisförmigen Objekten, die diese Pigmente enthalten. Je nachdem, wie weit diese Kriterien erfüllt sind, berechnet die Software eine Wahrscheinlichkeit, ob sich im aktuellen Bild ein Ball befindet und die Entfernung und den Winkel zum vermuteten Ball. Um vernünftige Werte zu erhalten ist es unabdingbar, die Software vorher zu kalibrieren[1]. Nichts destotrotz kann es zu false Positives kommen. Näheres zu den Ursachen von False Positives finden sich im Abschnitt 5 ab Seite 14.

2.3.2 Sonar-Partikelfilter

Es nutzt das im Roboter integrierte Sonarsystem, um anhand einer Karte die Position des Roboters im Raum zu bestimmen. Dazu bedienen wir uns seiner Klasse particleSet. Mit dem Konstruktor particleSet :: particleSet (**const char** ∗map_path, **const char** ∗wskTxT_path, **bool** use_ray_casting, **int** max_x, **int** min_x, **int** max_y, **int** min_y,**int** map_size, **int** map_res, **int** numofParticle, ArRobot ∗p3dx) erzeugen wir ein Objekt particleSet mit der Partikelmenge. Dabei übergeben wir ihm auch den Pfad zur Karte des Raumes und legen fest, ob wir Raycasting benutzen oder nicht, sowie die Auflösung und Koordinaten der Karte.

Am Partikelfilter haben wir ein paar Änderungen vorgenommen. Es gab keine Möglichkeit, von außerhalb des Filters auf die Koordinaten und die Ausrichtung des Roboters zuzugreifen. Dazu wurden die Methoden StartFilter () und findBestParticle () so erweitert, dass die aktuelle Position des Roboters und seine Ausrichtung als Zeiger auf das erste Element eines double-Arrays ([x,y,Theta, Partikelwsk.])

[1] Die genaue Vorgehensweise wird im Abschnitt 4 Nutzung ab Seite 12 beschrieben

zurückgegeben wird. Da dieses in findBestParticle() dynamisch mit new alloziiert wird, muss es, wenn es nicht mehr beötigt wird, mit **delete**[] gelöscht werden. Außerdem ist nun möglich die Anzahl der Sonare von außerhalb des Filters ohne neukompillieren einzustellen. Die Visualisierungen wurden aus Performancegründen entweder entfernt oder deaktiviert. Zudem wurde für eine einfachere Auswertung und weil die Partikelvisualisierung öfter zu Abstürzen unseres Programms geführt hat der Filter um eine Funktion erweitert, die, sofern dies beim Kompilieren der Software aktiviert wurde, bei jedem Filtern alle Partikel als Tripel (x,y,Partikelwsk.) in eine Datei schreibt. Die Datei hat dabei als Prefix eine Zahl im Namen, die bei 0 beginnend nach jedem Filtern um 1 erhöht wird. Passend dazu wurde ein Visualisierer geschrieben, der die Partikelmenge grafisch darstellt, auf den noch später eingegangen wird.

2.3.3 Funktionsweise des Clients

Der Client selber arbeitet folgendermaßen:
Beim Start wird die Konfiguration aus der Datei config.cfg eingelesen, wenn diese nicht gefunden wird, nutzt der Client die fest eingestellten Standardwerte. Die config.cfg hat folgende Struktur:

```
1  #Kommentarzeile (wird ignoriert)
2  Sonaranzahl als int
3  #Kommentarzeile (wird ignoriert)
4  COM-Port, an dem der Roboter angeschlossen ist, als String
5  #Kommentarzeile (wird ignoriert)
6  Schrittweite fuer die Anlerndrehungen als int in Grad
7  #Kommentarzeile (wird ignoriert)
8  Anzahl der 360Grad Drehungen als int
9  #Kommentarzeile (wird ignoriert)
10 Server-IP als String
11 #Kommentarzeile (wird ignoriert)
12 Pixel in der Karte pro mm als double
```

Listing 2.1: Aufbau der config.cfg

Ein Beispielkonfiguration für den Fahrstuhlvorraum vor dem Robotik-Labor im Braunschweiger Informatikzentrum wäre z. B.

```
1  #Anz Sensoren
2  8
3  #COM Port
4  COM5
5  #Schrittweite fuer die Anlerndrehungen
6  15
7  #Anzahl der 360Grad Drehungen
8  1
9  #Server-IP
10 134.169.36.241
11 #Pixel pro mm
12 0.05081
```

Listing 2.2: Ein Beispiel für eine komplette config.cfg

Die Standardwerte sind (16, COM3, 45, 0, 127.0.0.1, 1).

Danach wird die Verbindung zum Roboter hergestellt und der Partikelfilter mit 10000 Partikeln initialisiert und das erste Mal gefiltert. Danach erfolgt die konfigurierte Anzahl an Drehungen in der konfigurierten Schrittweite um die erste Positionierung zu verbessern. Nach jedem Drehschritt wird einmal gefiltert. Nach den Drehungen wird noch einmal gefiltert und der Rückgabewert als Startposition genommen. Erst jetzt wird die Verbindung zum Server hergestellt und die Roboterposition übermittelt, sowie ein Thread gestartet, der regelmäßig die Structs RobotInformation, PathInformation und SearchInformation mit dem Server abgleicht. Sie enthalten die zum Steuern des Clients, sowie Durchführung der Suche notwendigen Informationen. Nach Verbindungsaufbau wartet der Client bis der Server die Suche startet. Sobald die Suche startet, wird ein Thread zur Ballerkennung gestartet und der Client geht in seine Fahrschleife über, die erst endet, wenn der Ball gefunden wurde oder stoppt, wenn der Server die Suche pausiert oder beendet. Solange die Fahrschleife aktiv ist, wartet der Client auf den nächsten anzufahrenden Punkt vom Server, sobald er diesen erhalten hat fährt der Roboter den Punkt an. Dies geschieht schrittweise:
Zuerst wird die Richtung und der Drehwinkel der Drehung für eine Ausrichtung zum Punkt berechnet. Dazu wird der Winkel zwischen dem Ausrichtungsvektor des Roboters und dem Vektor vom Roboter zum Punkt berechnet und dann geprüft, ob der Vektor zwischen Roboterposition und dem Punkt um den ermittelten Winkel im Uhrzeigersinn oder gegen den Uhrzeigersinn von der Roboterausrichtung aus liegt um die Drehrichtung zu ermitteln. Dann wird der Roboter entprechend der ermittelten Werte zum Punkt hin gedreht. Dann wird, falls die Suche gerade pausiert, gewartet bis der Server die Pause beendet. Danach wird drivingTime $*$ driveLengthWeight ms gefahren, wobei drivingTime die Fahrtzeit pro Schritt ist und driveLengthWeight ein Anpassungsfaktor der Fahrzeit ist für den Fall, dass die Reststrecke in weniger als der Fahrzeit pro Schritt zurückgelegt werden kann. Dazu wird aus den gefahrenen Strecken in jedem Fahrtschritt und der jeweiligen Fahrtzeit die Geschwindigkeit in px pro ms berechnet und über die Schritte der exponentiell geglättete Mittelwert berechnet, woraus dann driveLengthWeight für die Reststrecke berechnet wird , wenn die zu klein für die normale Fahrzeit ist. Dann wird eine neue Ausrichtung zum Punkt berechnet und mit der aktuellen verglichen, verkürzt sich dadurch die Entfernung zum Ziel, wird der Roboter neu ausgerichtet, ansonsten wird die alte Ausrichtung beibehalten. Danach wird wieder eine Teilstrecke gefahren usw. bis sich dem Punkt auf einen Abstand unterhalb einer Toleranzschwelle angenähert wurde (delta_pos). Die Toleranzschwelle ist nötig, da aufgrund der Ungenaugkeiten des Sonar-Partikelfilters nicht immer eine genaues Erreichen der Zielposition gewährleistet werden kann. Auf die prinzipiellen Ungenauigkeiten des Partikelfilters wird näher im Abschnitt 5 Evaluation ab Seite 14 eingegangen. Danach wird gewartet, bis der Server das Erreichen des Punktes registriert hat und ein neuer, anzufahrender Punkt übermittelt wurde.

3 Erstellung der Software

In diesen Abschnitt beschreiben wir den Erstellungsprozess der Software. Dazu beschreiben wir erst die Struktur des Projektes, den Aufbau des Buildsystems für den Client und schließlich den Erstellungsprozess für Client und Server.

3.1 Aufbau des Projektbaums

Der aus dem SVN ausgecheckte Projektbaum ist wie folgt aufgebaut:

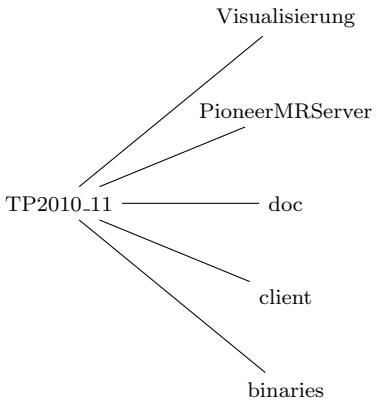

Abbildung 3.1: Projektbaum nach svn checkout

Da beim Client es nötig war, diverse Libaries des Institutes für Robotik und Prozessinformatik (IRP) der technischen Universität Braunschweig sowie den Sonar-Partikelfilter und die Ballerkennung zu integrieren, wurde dazu eine besondere Buildumgebung auf Basis von CMake geschaffen. Sie wird im Abschnitt 3.2 ab Seite 8 beschrieben.
Einen ersten Überblick über die Struktur des Ordners „client" gibt folgende Baumübersicht:

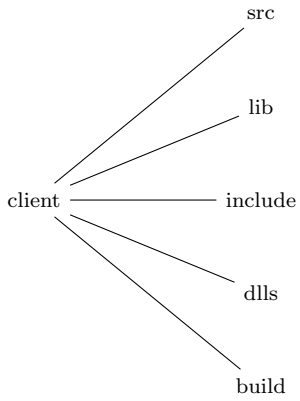

Abbildung 3.2: Projektbaum im Unterordner client

Der Ordner ,,build" enthält die durch CMake generierten VisualStudio Projekte und fertig gebauten ausführbaren Dateien und DLLs. Die Ordner ,,dlls" und ,,lib" enthalten zur Erzeugung und Ausführung der Projekte nötige externe *dll und *lib Dateien. In den Ordnern ,,include" und ,,src" sind schließlich die Quelltexte und Header der verwendeten Libaries sowie unseres Clients ,,p3dxSteuerung" enthalten.

Die Notwendigkeit einer gesondereten Buildinfrastruktur war beim Server (PioneerMRServer) nicht gegeben, da dieser nicht von den Institutsbibliotheken abhängt. Entsprechend reichte es dort, ein normales VisualStudio Projekt zu erstellen.

3.2 Integration bestehender Projekte in ein neues Buildsystem für den Client

3.2.1 Warum ein neues Buildsystem?

Die Ballerkennung und der Sonarpartikelfilter hängen von vielen Altprojekten ab. Das Beispiel für die Steuerung der Pioneer 3-DX hing zudem von der ARIA Bibliothek[1] ab. Bei den Altprojekten fanden sich auch zum Teil fest kodierte Pfade. Da auch die Konvertierung der Solutions in das VS 2010 Format meistens fehlschlug und auch ARIA nicht mit der mitgelieferten Solution unter VS 2010 gebaut werden konnte, haben wir uns für den Client für den Einsatz eines neuen Buildsystems entschlossen. Aufgrund der Einfacheit der Beschreibung der Buildvorgänge für die einzelnen zu integrierenden Projekte und dem Hinzufügen neuer Unterprojekte, sowie der Möglichkeit, von einem konkreten Buildsystem abhängig zu sein sowie existierender Erfahrungen mit dem System, fiel die Wahl auf CMake[2]. Bei CMake handelt es sich um ein open-source Metabuildsystem. Es erzeugt Build-Vorschriften für andere Buildsysteme (u.a. GNU make, nmake, msbuild). Somit wäre es bei Bedarf auch Möglich gewesen, auf eine andere Visual Studio Version umzusteigen ohne die Solutions zu konvertieren oder neu zu erstellen.

[1] http://robots.mobilerobots.com/wiki/ARIA
[2] http://www.cmake.org/

3.2.2 Integration der Projekte

Zuerst wurde eine passende Ordnerstrukur angelegt: include für Header-Dateien, src für die cpp-Dateien der einzelnen Projekt, bin für eventuell zum Linken benötigte Kompillate, deren Quellen nicht in den Build-Prozess integrierbar waren und build als Zielordner für erzeugte Buildfiles.

Als nächstes wurde für das Projekt im Ordner des Clients eine CMakeLists.txt-Datei angelegt:

```
1  cmake_minimum_required (VERSION 2.6)
2  project (TP_WS_2010)
3
4  include_directories (include)
5  include_directories (include/ControlExample)
6  include_directories (include/erob/)
7  include_directories (include/Balldetection)
8  include_directories (include/p3dxSteuerung)
9  include_directories (include/StochasticLib)
10 include_directories (include/PioneerMRClient)
11 include_directories (include/Aria)
12 include_directories (include/OccupancyGridMap)
13 include_directories (include/irpVideo/DirectShow)
14 link_directories (${TP_WS_2010_BINARY_DIR})
15 link_directories (${TP_WS_2010_SOURCE_DIR}/lib/)
16 add_subdirectory (src)
```

Listing 3.1: CMakeLists.txt für das Buildsystem des Clients

Diese definiert den Namen des Projektes (wichtig für Referenzen auf z.b. das Stammverzeichnis des Projektes oder das Buildverzeichnis, da die entprechenden Variablennamen mit dem Projektnamen + _ als Präfix versehen werden. Zudem wurden die include-Verzeichnisse und zum Linken relevante Verzeichnisse festgelegt. Danach wurde das src-Verzeichnis als Projektunterordner hinzugefügt. Dadurch werden die Anweisungen in der CMakeLists.txt des Unterordners auch ausgeführt. Dies ermöglicht es, für die einzelnen Unterprojekte jeweils eine eigene kleine CMakeLists.txt zu verwenden.

Die CMakeLists.txt bindet nun alle Sourceordner der Unterprojekte ein:

```
1  %\begin{lstlisting}
2  cmake_minimum_required (VERSION 2.6)
3
4  add_subdirectory (irpUtils)
5  add_subdirectory (irpMath)
6  add_subdirectory (irpImage)
7  add_subdirectory (irpVideo)
8  add_subdirectory (irpFeatureExtraction)
9  add_subdirectory (alglib)
10 add_subdirectory (DistanceMap)
11 add_subdirectory (irpCamera)
12 add_subdirectory (irpV3d)
13 add_subdirectory (StochasticLib)
14 add_subdirectory (Visualization)
15 add_subdirectory (CamCalib)
16 add_subdirectory (Balldetection)
17 add_subdirectory (mathtest)
18 add_subdirectory (SonarParticleFilter)
19 add_subdirectory (PioneerMRClient)
```

```
20 add_subdirectory (OccupancyGridMap)
21 add_subdirectory (Balldetection_Test)
22 add_subdirectory (Aria)
23 add_subdirectory (p3dxSteuerung)
24 add_subdirectory (p3dxSteuerung_threaded)
25 add_subdirectory (calibrate)
26 #add_subdirectory (ControlExample)
27 add_subdirectory (utils)
28 add_subdirectory (kinect)
29 add_subdirectory (networkTest)
```

Listing 3.2: CMakeLists.txt für den Sourceordner

Bei den Unterprojekten gibt es zwei Arten: Bibliotheken und Anwendungen.

Die CMakeList.txt einer Bibliothek sieht wie folgt aus:

```
1 %\begin{lstlisting}
2 cmake_minimum_required (VERSION 2.6)
3
4 file(GLOB src "*.cpp")
5 file(GLOB includes "../../include/irpCamera/*.h")
6
7 add_library (irpCamera ${src} ${includes})
```

Listing 3.3: CMakeLists.txt einer Bibliothek am Beispiel irpCamera

Die CMakeList.txt einer Anwendung sieht so aus:

```
1 cmake_minimum_required (VERSION 2.6)
2
3 file(GLOB src "*.cpp")
4 file(GLOB includes "../../include/p3dxSteuerung/*.h")
5 add_executable (p3dxSteuerung_threaded ${src} ${includes})
6 target_link_libraries (p3dxSteuerung_threaded irpUtils MINPACK fftw rfftw
7 DirectShow irpFeatureExtraction irpMath  alglib irpImage irpCamera irpVideo
8  irpV3d glew CamCalib Visualization StochasticLib OccupancyGridMap
9 SonarParticleFilter ws2_32 winmm advapi32 Aria DistanceMap Balldetection)
```

Listing 3.4: CMakeLists.txt des Clients

Zuerst werden alle cpp-Dateien zur Variablen src hinzugfügt, danach die Includes zur Variablen include. Je nachdem, ob es sich um eine Bibliothek oder Anwendung handelt, wird diese mit add_library(name dateien) oder add_executable(name dateien) hinzugefügt. Die Include-Dateien müssen dabei aber eigentlich nicht explizit hinzugefügt werden, dies geschieht nur, damit sie auch in einer VS Solution in der Dateiliste auftauchen. Bei Anwendungen können nun mit target_link_libraries(exename libraries) noch die zu linkenden Bibliotheken angegeben werden.

Es gab ein paar Probleme bei der Integration: zum einen mussten zuerst die Abhängigkeiten zwischen den Unterprojekten ermittelt werden, dies geschah mit Hilfe der originalen VS Solutions sowie durch Ausprobieren. Zum anderen gab es manchmal Dateien, die zwar in den Quellen lagen, jedoch in nicht in den Solutions auftauchten. Diese konnten den Buildprozess stören und mussten daher entfernt werden TODO: Beispiel raussuchen, wenn noch überhaupt möglich. Außerdem waren an ein paar Stellen kleinere

Änderungen am Quellcode nötig, um ihn mit MSVC 2010 oder der Struktur unseres include-Ordners kompatibel zu machen.

3.2.3 Hinzufügen eines neuen Unterprojektes

Das Hinzufügen eines neuen Unterprojektes ist relativ einfach. Zuerst wird ein Unterverzeichnis für das Projekt im include-Ordner und eines im src-Ordner angelegt. Dann kopiert man die includes in den neuen Unterordner im include-Ordner. Dabei ist darauf zu achten, dass die Pfade in #include-Direktiven relative zum include-Ordner sein sollten. Danach kopiert man die cpp-Dateien in den neuen Unterordner im src-Verzeichnis.
Danach erstellt man in diesem Ordner eine CMakeLists.txt ähnlich wie oben, jenachdem ob ex sich um eine Anwendung oder Bibliothek handelt. Man fügt zuerst die cpp-Dateien und Includes hinzu, fügt dann mit add_library oder add_executable Buildtargets hinzu und gibt eventuell noch zu linkende Bibliotheken bei Anwendungen an mit target_link_libraries.

3.3 Erstellung von Client und Server mit Visual Studio 2010

Da der Client mit Hilfe des im vorherigenen Abschnitts auf CMake basierenden Buildystems erzeugt wird, sind hier mehr Schritte als beim Server notwendig:

1. CMake für Windows (Version min. 2.6) herunterladen und installieren [3]
2. CMake GUI starten
3. Unter "Where is the source code"den Pfad des Client-Verzeichnisses eintragen und unter "Where to build the binaries"den Pfad zu dem Verzeichnis, wo die Solution für den Client erstellt werden soll
4. Auf "Configure"klicke, "Visual Studio 2010ünd Üse default native compilersäuswählen, dann auf Finish klicken. Auf das Ende des Einrichtens warten.
5. Auf "Generate"klicken
6. Im ausgewählten Zielverzeichnis befindet sich nun eine Visual Studio 2010 Solution (TP_WS_2010.sln).
7. Außerdem befindet sich im Unterordner „PioneerMRServer" im Projektbaum eine Solution für den Server (PioneerMRServer.sln). Beide Solutions können ganz normal mit Visual Studio 2010 geöffnet und gebaut werden (F7 oder STRG-Alt-F7).

[3] http://www.cmake.org/

4 Nutzung der Software

4.1 Client

4.1.1 Kalibrierung der Balldetection

Die Ballerkennung muss zunächst kalibriert werden, da ansonsten keine brauchbaren Ergebnisse erzielt werden können. Dies ist deshalb nötig, da die Ausrichtung der auf dem Roboter montierten Webcam entscheidenden Einfluss auf die von der Ballerkennung ermittelten Werte hat. Zur VisualStudio Solution ,,TP2010/11" gehört auch ein Unterprojekt calib. Nach Erstellung der ausführbaren Datei calib.exe ist diese an folgende Stelle des Projektbaums zu kopieren: `\client\src\calibrate\`
Für die folgenden Schritte werden dann ein Schachbrett sowie ein Maßband oder Ähnliches benötigt. Danach geht man folgendermaßen vor:

- Man stellt das Schachbrett und den Roboter in einen Abstand von ca 1 Meter voneinander entfernt so auf, dass die Kamera auf die Mitte des Schachbrettes zeigt: TODO: bild einfügen
- Danach misst man mit dem Maßband die Entfernung zwischen Roboter und Schachbrett, sowie den Abstand vom Schachbrett zum Boden: TODO: bild einfügen
- Anschließend startet man die Batchdatei `calibrate.cmd` im Ordner `\client\src\calibrate\`. Nun muss man noch die Kameranummer, sowie die im vorherigen Schritt ermittelten Entfernungen in mm angeben und die Kalibrierung startet: TODO: bild einfügen
- Die Kalibrierung ist erfolgreich, wenn folgendes Muster im Kamerabild angezeigt wird, dann ist im Batch-Skript die Ausführung des Kalibiertools mit ¡STRG¿-C zu unterbrechen. Auf die Nachfrage, ob auch die Batchdatei unterbrochen werden soll, gibt man ,,n" ein.
- Danach sollte ein neuer Ordner im aktuellen Verzeichnis mit den Kalibrierdaten sein.

4.1.2 Einrichtung des Clients

Der Client wird über die Datei `config.cfg` eingerichtet. Eine Erklärung und ein Beispiel finden sich im Abschnitt 2.3.3 auf Seite 5. Die Datei muss sich im gleichen Verzeichnis wie die ausführbare Datei `p3dxSteuerung.exe` befinden. Ausserdem benötigt der Client die Daten der Kalibrierung und Karten für die Lokalisierung. Idealerweise erzeugt man sich für die *exe-Datei und anderen benötigten Dateien

ein eigenes Verzeichnis, wo alle benötigten Dateien rein kopiert werden. Hierzu empfiehlt sich folgender Aufbau des Laufzeitverzeichnisses:

4.1.3 Benutzung des Clients

5 Evaluation

5.1 Visualisierung der Partikelmengen des Partikelfilters

Da die im Partikelfilter integrierte Visualisierung der Partikelmenge nicht immer funktionierte und es auch keine direkte SPeichermöglichkeit der Bilder gab, haben wir den Partikelfilter, so erweitert, dass die Partikelmenge in eine Datei geschrieben werden kann (2.3.2) und passend dazu ein Skript (in Python unter Benutzung von pygame geschrieben), welches die Partikel anhand der Daten aus einer Partikelmengendatei über ein Bild legt. Dieses ist im Ordner Visualisierung im SVN-Repository unter dem Namen visualize.py zu finden.

5.1.1 HOWTO: Installieren der Abhängigkeiten des Visualisierungsskriptes und Benutzung dessen

- Python 3.2 herunterladen und installieren[1]
- pygame 1.9.2a0 für Python 3.2 installieren [2]
- Python 3.2 zum PATH hinzufügen
- cmd/Eingabeaufforderung öffnen
- In den Ordner Visualisierung des Repositories wechseln
- Das Visualisierungsskript kann nun folgendermaßen benutzt werden:
  python visualize.py (Dateiname des Bildes) (Dateiname der Partikelmengendatei)
  z.b. python visualize.py OccuMap.bmp visual.log
- Das Ausgabebild hat dann den Namen V_(Dateiname der Partikelmengendatei)_(Dateiname des Bildes)

[1] http://python.org/ftp/python/3.2.2/python-3.2.2.msi
[2] http://pygame.org/ftp/pygame-1.9.2a0.win32-py3.2.msi