Allgemeine Hinweise zum Praktikum

Bei der Anmeldung zur Veranstaltung wurde jede Arbeitsgruppe einer Rechnerzeit zugeordnet. Diese gelten für die Rechner des Fachgebiets in den Laborräumen EN258 und EN266a zur Programmierung der Robotersteuerung. Für den gesamten Übungsbetrieb ist es egal, auf welchem Rechner gearbeitet wird. Die Ausführungsrechner für die von Ihnen erstellten Programme stehen im Labor EN266a.

Alle nötigen Dateien stehen in `/home/pdv/pdv/lehre/ees/`. Dateien zu Aufgabe Nummer *X* stehen im Verzeichnis `/home/pdv/pdv/lehre/ees/zu_a`*X*.

Zur Lösung der Programmieraufgaben gehören

- die **Vorführung** des funktionierenden Programms
- ab Aufgabe 2 die **Modulstruktur** des Programms in Form einer Graphik mit den wichtigsten Abhängigkeiten, insbesondere die **Prozeßkommunikation** ab Aufgabe 3
- und ein **kommentiertes Programm**, das per **E-Mail** dem Betreuer gesendet wird.

Initialisieren der Arbeitsumgebung (siehe auch Großübung 1)

Zu Beginn jeder Sitzung muß die Umgebung initialisiert werden mit dem Befehl `source /home/pdv/pdv/lehre/ees/init` (in jedem X-Terminal, alternativ kann dieser Befehl auch in `~/.bashrc` bzw. `~/.cshrc` geschrieben werden). Achtung: je nach verwendeter login-Shell muß entweder `init.bash` (default) oder `init.tcsh` verwendet werden. Wenn danach `which makelib` als Ergebnis `/home/pdv/pdv/lehre/ees/makelib` ausgibt, hat es funktioniert.

Verzeichnisse müssen zur Arbeit mit Ada zu einer Ada-Library modifiziert werden mit dem Befehl `makelib`. Dadurch werden verschiedene Dateien erzeugt, die nicht von Hand zu verändern sind. `makelib` läuft erfolgreich nur auf dem Rechner `bolero`, also zur Ausführung von `makelib` dort anmelden.

Programmentwicklung (siehe auch Großübung 1)

Ada-Programm schreiben (`name_s.a` : specification, `name_b.a` : body)

Programm übersetzen (auf richtige Reihenfolge achten) `ada name_s.a` ...

Programm binden: `a.ld` <Name Hauptprogramm> `-o` <Programmname>
Bsp.: `a.ld robot_main -o move.vox`

Programm auf Echtzeitrechner starten: `start blei move.vox` (alternativ: `platin`)

Aufgabe 1 — Abfrage der Endabschalter

Lernziel: Eingewöhnung in die Entwicklungsumgebung, Programmierung einer Schnittstelle, Ausgabefunktionen

Unterlagen: Ada Language Reference Manual
Schnittstellenbeschreibung
Vorgaben im Verzeichnis `/home/pdv/pdv/lehre/ees/zu_a1`
Unterlagen zur ersten Großübung

Aufgabenstellung

Schreiben Sie ein Programm, das den Zustand der Endabschalter des Roboters einliest und auf dem Terminal ausgibt. Die Ausgabe muß einmal zum Programmstart erfolgen (initiale Schalterstellungen) und mindestens immer dann erfolgen, wenn einer der Schalter betätigt wird. Das Programm soll terminieren, wenn jeder Endabschalter mindestens einmal **um**geschaltet worden ist. Die Endabschalter werden mit Polling abgefragt.

Statt der Endabschalter der Roboter ist für die erste Aufgabe an die Eingangsleitungen der Parallelschnittstelle eine Black Box angeschlossen, an der die Schalter manuell betätigt werden können.

Alle notwendigen Initialisierungen der Steuerregister werden bereits durch `PARALLEL_IO` durchgeführt. Nur noch der Zugriff auf die Datenregister ist zu realisieren. Dazu müssen entsprechende Typen und Variablen definiert werden und letztere an die richtigen Adressen gebunden werden.

Ein Beispielprogramm wird ausführlich in der ersten Großübung (Termin siehe Organisationsunterlagen) besprochen.

Aufgabe 2 — Ansteuerung der Robotermotore

Lernziel: Ansteuerung einzelner Motoren mit Schritten, Einbindung von digitaler Sensorik nach der Methode des wiederholten Abfragens, Eingabefunktionen

Unterlagen: Ada Language Reference Manual
Schnittstellenbeschreibung, Roboteransteuerung
Vorgaben im Verzeichnis `/home/pdv/pdv/lehre/ees/zu_a2`

Aufgabenstellung

Schreiben Sie ein Programm, das einzelne Gelenke um eine interaktiv eingegebene Anzahl von Schritten bewegt. Das Programm darf Schritte nicht schneller ausgeben, als der Roboter sie ausführen kann, da sonst die korrekte Überwachung der Endabschalter nicht möglich ist. Zum Verzögern des Programms dürfen `delay`-Anweisungen nicht verwendet werden sondern nur Zählschleifen, siehe Hinweis unten. Das Programm soll terminieren, wenn die Taste „E“ betätigt wird. Die Ist-Positionen aller Gelenke sind auf dem Bildschirm auszugeben.

Programmieren Sie hierzu eine Prozedur `GELENKSCHRITTE`, die ein Gelenk um eine bestimmte Schrittanzahl bewegt. Der Prozedur wird der Name des Gelenks und die Anzahl der gewünschten Schritte vorzeichenbehaftet übergeben. Es wird zu einer Zeit immer nur ein Gelenk bewegt. Die Prozedur darf eine Bewegung über die Begrenzungen hinaus nicht zulassen. Mit Ihrem Programm müssen alle Gelenke des Roboters bewegt werden können.

Es soll eine globale Variable `IST_POSITION(GELENK)` verwendet werden, in der die derzeitigen Positionen der Gelenke in Schritten von der Nullstellung weg gespeichert werden. Zur Initialisierung dieser globalen Variablen dient die vorgegebene Prozedur `NEST`, die die Funktionalität des NEST-Befehls der Mitsubishi-Teachbox nachbildet, d. h., Gelenke einzeln in ihre Nullposition fährt. Zu Programmbeginn und -ende müssen alle Gelenke einzeln genestet werden. Bitte beachten Sie dabei folgende Reihenfolge, in der die Gelenke zu nesten sind:

1. Schultergelenk
2. Handneigegelenk
3. Handdrehgelenk
4. Ellbogengelenk
5. Körpergelenk

Nach Ausführung der Prozedur `NEST` für jedes Gelenk ist den `IST_POSITION`en der einzelnen Gelenke der Wert 0 zuzuweisen.

Bei der Berührung eines Endabschalters während einer Bewegung muß die Bewegung des angetriebenen Gelenks sofort abgebrochen werden. Dem Bediener ist dann eine Auswahlmöglichkeit zum weiteren Betrieb des Systems anzuzeigen.

Bitte beachten Sie folgende Besonderheiten der Endabschalter:

- Fährt ein Gelenk aus seiner Nullage heraus, ist der Endabschalter noch ca. 10 Schritte gedrückt. Dies ist kein Fehler, sondern eine Eigenschaft mechanischer Schalter und nennt sich Hysterese. Die Nullage ist definiert durch den ersten Punkt bei der Bewegung auf den Endabschalter zu, bei der der Schalter gedrückt ist. Gedrückte Endabschalter müssen demnach abhängig von der aktuellen Bewegungsrichtung ignoriert oder beachtet werden. Außerdem ist es möglich, daß durch mechanische Toleranzen des Roboters Endabschalter nicht bewegter Gelenke kurzzeitig ihren Zustand ändern.

- Das Handdrehgelenk hat folgende Eigenschaften:
  - Es darf über den Endabschalter hinaus bewegt werden, der Endabschalter hat nur eine Kalibrierfunktion,
  - der Endabschalter kann nur bei Handneigung = 0 gedrückt werden, und
  - der Endabschalter kann aus zwei Richtungen gedrückt werden.

  Natürlich ist nur diejenige dieser Stellen als Nullage definiert, die beim Nesten eingenommen wird, und nur hier darf die Ist-Position zurückgesetzt werden.

In der Testphase des Programm dürfen nicht mehr als ±100 Schritte eingegeben werden. Diese Beschränkung ist notwendig, um mechanische Überlastungen des Roboters durch abrupte Bewegungen zu vermeiden. Zur Vorführung der Aufgaben dürfen dann, wenn das Programm fehlerfrei arbeitet, ±500 Schritte eingegeben werden.

Damit die einzelnen Schritte der Motoren nicht zu schnell ausgeführt werden, ist es notwendig, das Programm zu verlangsamen. Dies darf nicht mit `delay`-Anweisungen erfolgen sondern muß mit einer Zählschleife realisiert werden. Folgende Schleife hat sich für erste Versuche bewährt:

```
for i in 0 .. 50000 loop
     k := k + 1;
end loop;
```

Bei dieser und allen folgenden Aufgaben werden Eingaben vom Benutzer eingelesen. Das Programm darf bei Falscheingaben nicht abstürzen!

Da ein technisches System von Ihnen programmiert und betrieben wird, obliegt Ihnen höchste Sorgfaltspflicht bei der Programmierung und dem Betrieb der Roboter. Bitte stellen Sie sicher, daß alle Fehlerfälle erkannt und abgefangen werden. Hierzu ist es erforderlich, den Bediener durch entsprechende Meldungen auf Fehlerzustände hinzuweisen und ggf. Eingaben zum weiteren Betrieb abzufragen.

Und ganz selbstverständlich ist, daß eine Hand auf dem Notausschalter ruht, damit jederzeit der Roboter abgeschaltet werden kann.

Aufgabe 3 — Verarbeitung digitaler Signale

Lernziel: Programmieren mit Zeiten

Unterlagen: Ada Language Reference Manual
Vorlesungsunterlagen

Aufgabenstellung

Entwickelt einen einfachen Dekodierer für Morsezeichen. Das Programm muß mittels zweier nebenläufiger Tasks realisiert werden.

Es steht wieder die Endabschalter-Textbox zur Verfügung, die an die Echtzeitrechner angeschlossen ist. Dort, wo in Aufgabe 1 der Endabschalter 0 abzufragen war, ist jetzt ein elektronischer Morsegeber angeschlossen. (Bitte beachten: Der Morsegeber ist parallel zu dem Endabschalter angeschlossen. Damit nicht ein Dauerträger empfangen wird, muß Schalter 0 offen sein, also nach oben geschaltet. Ferner ist die Polarität des Anschlusses zu beachten: Der rote Stecker kommt in die rote Buchse, der blaue Stecker in die blaue Buchse) Die Lösung der Aufgabe 1 kann zum Erkennen der Morsetaste nach einigen Veränderungen weiterverwendet werden.

Im Verzeichnis `/home/pdv/pdv/lehre/ees/zu_a3` geben wir in der Datei `morse.txt` die Morsezeichen vor. Wer will, kann zur Zeichendefinition und Ausgabe der Zeichen die Vorgabe aus `morse_def.a` verwenden, es können aber auch eigene Routinen entwickelt werden. Ferner wird die Bibliothek `Calendar` benötigt, also sollte in Eurem Programm irgendwo `with Calendar; use Calendar;` stehen. `morse_def.a` liegt nicht in der `eeslib/` als Bibliothek vor, sondern muß in Euer Verzeichnis kopiert und dort kompiliert werden, wenn es verwendet werden soll.

Der Dekodierer soll mittels **zweier Tasks** realisiert werden: Eine Task `lesen` soll den Schalterzustand überwachen und registrieren, wie lange die Morsetaste gedrückt bzw. offen ist. Sobald eine Pause registriert wird, die dem Abstand zwischen zwei Zeichen (Buchstaben, Satzzeichen, Verkehrszeichen) entspricht, sollen die Zeiten (entsprechen den Punkten, Strichen und Pausen) einer zweiten Task `auswerten` übergeben werden, die dann ermittelt, welches Zeichen gelesen wurde.

Damit der Dekodierer mit verschiedenen Gebegeschwindigkeiten zurechtkommt, muß er zunächst kalibriert werden. Hierzu sind eine Reihe „v“ (`...-`) zu geben. Aus dem Kalibriervorgang resultieren dann die Werte für die Länge der Punkte bzw. Striche. Nach dem Kalibrieren sollen dann die gemorsten Zeichen entschlüsselt und auf dem Monitor ausgegeben werden. Zeichen, die nicht erkannt wurden, sollen mit einem Bindestrich - dargestellt werden. Macht der Funker eine Pause, die länger als die normale Pause zwischen zwei Zeichen ist, so soll einmal ein Freizeichen (*space*) ausgegeben werden. Das Kalibrieren kann in der Task `lesen` erfolgen.

Folgende weitere Randbedingungen sollen gelten: Das Programm soll terminieren, wenn ein Dauerträger von 60 Sekunden gegeben wird oder der Funker mehr als 60 Sekunden Pause macht. Ferner soll das Programm terminieren, wenn $\overline{\text{SK}}$ (`...-.-`) oder $\overline{\text{AR}}$(`.-.-.`)

gegeben wurde. Bei diesen Zeichen handelt es sich wie bei den Zeichen $\overline{\text{KN}}$, $\overline{\text{DN}}$, $\overline{\text{BT}}$, $\overline{\text{AAA}}$, $\overline{\text{MIM}}$, $\overline{\text{IMI}}$ und $\overline{\text{ERR}}$ um Satz- bzw. Verkehrszeichen, die aus mehreren Buchstaben zusammengesetzt sind. Zur besseren Unterscheidung werden diese Zeichen mit einem Überstrich versehen, wenn die Darstellung in Buchstaben gewählt wird. Für diese Aufgabe ist dies aber nicht möglich, bitte verwendet die ganz normale Darstellung `/ . ,` ?, $\overline{\text{ERR}}$ wird als ~ geschrieben.

Zur Theorie der Morsezeichen: Die Basiseinheit ist der Punkt, daraus wird das Verhältnis Punkt : Strich : Elementabstand : Zeichenabstand : Wortabstand mit 1 : 3 : 1 : 3 : 7 abgeleitet. Allerdings ist manuelles Geben mit einer klassischen Taste nicht allzu genau, daher sind Toleranzen vorzusehen, z.B. kann der Punkt durchaus auch mal 1,5 Punkte lang werden, der Strich aber nur 2,5 Punkte lang sein. Ebenso kann die Länge der Pausen zwischen den Elementen etwas schwanken.

Damit Euer Dekodierer nicht zu empfindlich auf schwankende Gebegeschwindigkeiten reagiert, sind Fristen für die Punkte bzw. Striche vorzusehen. Im einfachsten Fall und bei niedrigen Gebegeschwindigkeiten kann es ausreichen, Punkte und Striche zu unterscheiden, indem alle Werte kleiner einer Grenze als Punkte und alle Zeichen größer dieser Grenze als Striche interpretiert werden. Beachtet hierbei, daß die Auflösung der Zeit, die bei Verwendung der Standarduhr erreicht werden kann, bei 0,01 Sekunden liegt.

agbp es vy 73

(Gruß der Morse-Funker: Always good brass pounding and with very best compliments)

Hinweise zum elektronischen Morsegeber:

Groß- und Kleinschreibung werden nicht unterschieden. Es können die Ziffern 0 bis 9, die Buchstaben a-z, ä, ö, und ü sowie die Satzzeichen Komma, Punkt, Trennung, Schrägstrich und Fragezeichen auf der Tastatur eingegeben werden. Das Fragezeichen ist ohne SHIFT erreichbar, der Schrägstrich liegt auf dem + (also rechts neben dem ü). Ferner sind die Tasten F1 mit „vvv“ und die F2 mit „paris“ belegt. Tasten, die der Morsegeber nicht interpretiert, erzeugen keine Ausgaben und keine Fehler. Mit ALT-A wird $\overline{\text{AR}}$, mit ALT-S wird $\overline{\text{SK}}$ eingegeben.

Der Morsegeber puffert einige Zeichen, sollte an der Tastatur schneller eingegeben werden als gemorst wird. Mit dem Tabulator (TAB) kann die Ausgabe jederzeit abgebrochen werden.

Der Morsegeber verfügt über 2 Geschwindigkeiten, die mit der ENTER-Taste umgeschaltet werden. Bei Geschwindigkeit 2 leuchtet die CAPS-LOCK-LED der Tastatur. Geschwindigkeit 1 ist langsam, Geschwindigkeit 2 ist schnell. Euer Programm soll beide Geschwindigkeiten beherrschen, wobei ein Neustart des Programms erlaubt ist.

Die DEL-Taste erzeugt einen Dauerton, zum Abschalten noch einmal DEL drücken. Die ESC-Taste wird **nicht** benutzt.

Durch Aus- und Wiederanschalten erfolgt ein Reset des Morsegebers auf seine Standardwerte. Bitte geht pfleglich mit den Morsegebern um!

Aufgabe 4 — Kugelfallversuch

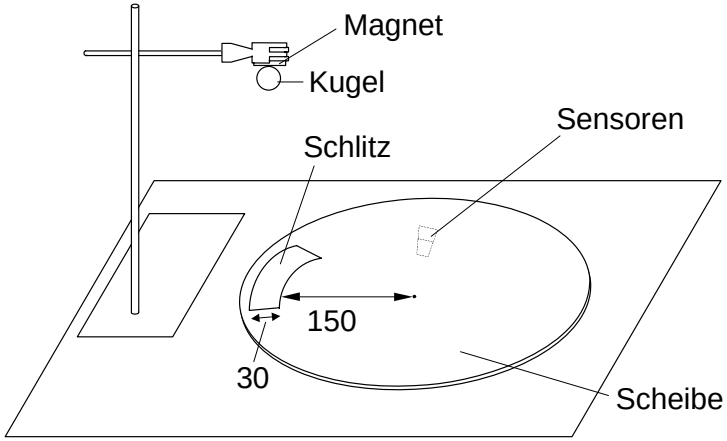
Lernziel: Echtzeitprogrammierung mit Ada

Unterlagen: Ada Language Reference Manual
Vorgaben im Verzeichnis /home/pdv/pdv/lehre/ees/zu_a4

Aufgabenstellung

Es soll ein Programm geschrieben werden, welches eine magnetisch gehaltene Kugel rechtzeitig auslöst, so daß diese durch den Schlitz einer sich drehenden Scheibe fällt.

Versuchsaufbau



Die Scheibe hat einen Durchmesser von 400 mm und ist 17 mm dick; der Durchmesser der Kugel beträgt 12 mm. Der Schlitz ist 30 mm breit und 150 mm vom Scheibenmittelpunkt entfernt. Sein Öffnungswinkel beträgt 39 Grad. Fallhöhe h (gemessen von der Kugelunterseite bis zur Scheibenoberseite) und Drehgeschwindigkeit sowie -richtung der Scheibe können von Versuch zu Versuch variiert werden.

Theoretischer Aufgabenteil

Stellen Sie die Formel für den richtigen Auslösezeitpunkt in Abhängigkeit von Fallhöhe, Umdrehungszeit, Scheibendicke und Kugeldurchmesser auf! Lassen Sie in Ihre Berechnungen eine Sicherheitsreserve mit eingehen, so daß nicht die gesamte Schlitzlänge ausgenutzt wird.

Berechnen Sie die minimal mögliche Umdrehungszeit der Scheibe (die der maximal möglichen Drehgeschwindigkeit entspricht), um noch mit einer Kugel (Durchmesser D_k) aus der Fallhöhe h durch den Schlitz der Scheibe zu treffen.

Die Ergebnisse Ihrer Berechnungen sind schriftlich als Teil der Aufgabenlösung abzugeben.

Hinweise**:**

- Der Luftwiderstand kann vernachlässigt werden.
- Erdbeschleunigung: g = 9,81 m/sec^2
- Es kann der Einfachheit halber davon ausgegangen werden, daß sich die Scheibe mit konstanter Geschwindigkeit bewegt. Wer Lust hat, kann als schönere Lösung implementieren, daß die die Scheibe verlangsamende Reibung (zumindest näherungsweise) mit beachtet wird.
- Vom Zeitpunkt des Abschaltens des Magneten bis zum Beginn des Kugelfalls vergeht eine kurze Zeit, die ebenfalls vernachlässigt werden kann.

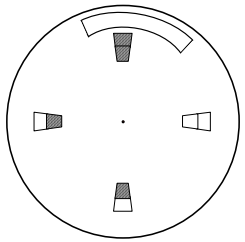
Praktischer Aufgabenteil

Schreiben Sie ein Ada-Programm, welches die Kugel so auslöst, daß diese ohne anzustoßen durch den Schlitz der sich drehenden Scheibe fällt. Die Fallhöhe muß zu Beginn des Versuchs gemessen und vom Programm eingelesen werden. Berechnen und zeigen Sie die sich daraus ergebende minimale Umdrehungszeit! Danach wird die Scheibe in Drehung versetzt. Ist eine konstante Geschwindigkeit erreicht, muß die Kugel zum vorher berechneten Zeitpunkt ausgelöst werden und durch den Schlitz fallen. Das Programm soll eine entsprechende Fehlermeldung ausgeben, wenn die Umdrehungszeit zu gering ist. Danach kann die Scheibe von Hand gebremst werden oder das Programm einfach abwarten, bis durch die Reibung die Geschwindigkeit gering genug ist. Beachten Sie außerdem, daß die Kugel zu einem geeigneten Zeitpunkt am Magnet befestigt werden muß.

Strukturieren Sie Ihr Programm übersichtlich! Beispielsweise können die Funktionen für Sensoren und Magnet in einem Paket implementiert werden, welches dann vom Hauptprogramm aus importiert wird.

Hinweise zur Hardware

Der Versuchsaufbau ist an einem der Echtzeitrechner angeschlossen. Zur Verfügung stehen zwei optische Sensoren zur Lage- und Drehrichtungserkennung der Scheibe.



Über den Sensoren sind an der Unterseite der Scheibe Markierungen in Abständen von 90 Grad angebracht (siehe Abbildung). Wie im Bild skizziert, sind die Markierungen so angebracht, dass sich ein Rand der Doppelmarkierung genau in der Mitte des Schlitzes befindet.

Der aktuelle Sensorzustand kann über Baustein X, Port A, (Ada-) Bits 7 und 6 (Eingabe) abgefragt werden. Befindet sich eine in der Skizze dunkel dargestellte Markierung über dem Sensor, ist der ausgelesene Wert 0.

Bit 5 (Ausgabe) von Port A steuert den Haltemagnet. Er ist eingeschaltet, wenn das Bit den Wert 1 hat. Zur einfacheren Fehlersuche zeigen die drei Leuchtdioden des Steuergerätes den aktuellen Zustand der beiden Sensoren sowie des Magneten an. Unabhängig vom Steuerrechner kann der Magnet aktiviert werden, indem der Schalter am Steuergerät in die obere Position gebracht wird. Mit dem Drehknopf kann die Spannung am Magnet und damit die Haltekraft eingestellt werden.

Für die korrekte Übertragung von Sensor- und Magnetsignalen zwischen Echtzeitrechner und Steuergerät muß der verwendete Schnittstellenbaustein entsprechend programmiert werden! Beachten Sie in diesem Zusammenhang u. a. die Richtung der einzelnen Port-Bits (Ein- bzw. Ausgabe).

Für die notwendigen Zeitmessungen steht ein Timer-Baustein zur Verfügung. Die zu seiner Ansteuerung notwendigen Funktionen sind bereits im Paket TIMER realisiert, welches nur aus dem Vorgabenverzeichnis kopiert und übersetzt werden muß.

Verwendbare Funktionen:

`procedure TIMER_STARTEN;` Setzt den Zähler auf Null und startet den Timer

`procedure TIMER_STOPPEN;` Hält den Zähler an

`function TIMER_LESEN return FLOAT;` Gibt gemessene Zeit zwischen `TIMER_STARTEN` und `TIMER_STOPPEN` in Millisekunden zurück

`procedure WARTEN (ZEIT : in FLOAT);` Wartet angegebene Zeit in Millisekunden

Zum Schluß noch ein allgemeiner Hinweis: Bitte behandeln Sie den vorhandenen Aufbau so, daß alle Gruppen die Möglichkeit haben, die Aufgabe zu bearbeiten! Seien Sie bitte vorsichtig bei den Kugelfallversuchen und setzen Sie die Scheibe nicht in zu schnelle Drehung, um Unfälle zu vermeiden.