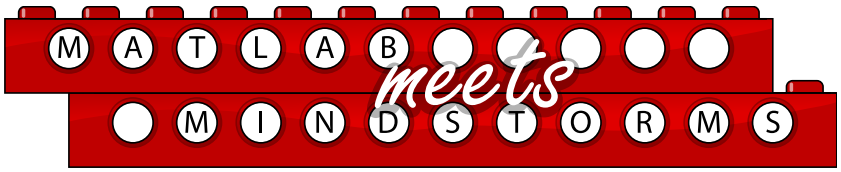
Projekt der Elektrotechnik und Informationstechnik



5. Projektversuch

- Lichtsensor -

15. Januar 2020

Inhaltsverzeichnis

1 Einleitung

Der Lichtsensor des LEGO Mindstorms Set ist ein Helligkeits- und Farbsensor. Er kann eingesetzt werden, um sieben verschiedene Farben zu unterscheiden oder um Helligkeitslevel, entweder vom Umgebungslicht oder einer Oberfläche, zu detektieren.

In diesem Versuch lernen Sie diesen Sensor näher kennen und werden darüber hinaus mit dem Konzept der Callback-Funktionen von MATLAB vertraut gemacht. Solche Callback-Funktionen werden in MATLAB beispielsweise bei der Verwendung von Timern und der Programmierung einer grafischen Benutzeroberfläche (GUI) eingesetzt. Timer können dazu verwendet werden, um bestimmte Funktionen periodisch (oder verzögert) auszuführen, z.B. beim periodischen Auslesen eines Sensors.

Mithilfe des App Designers können in MATLAB einfache GUIs erstellt werden. Dies ist vor allem dann sinnvoll, wenn komplexe Funktionen auf einfache Weise zugänglich gemacht oder demonstriert werden sollen.

Nach der Bearbeitung dieses Versuchs sollten Sie in der Lage sein, Timer korrekt zu verwenden und einfache GUIs zu erstellen.

2 Grundlagen

Im Folgenden wird der Lichtsensor beschrieben und die Funktionen eingeführt, mit denen der Sensor angesteuert werden kann. Außerdem werden Sie mit den MATLAB-Grundlagen vertraut gemacht, die in diesem Versuch benötigt werden. Dazu zählen der Umgang mit Timern sowie die Erstellung einer einfachen GUI in MATLAB. Sie erlernen das für diese beiden Themen grundlegende Konzept der Callback-Funktionen.

2.1 Mindstorms EV3 Lichtsensor

Der in diesem Projekt verwendete Lichtsensor, zusehen in Abbildung 1, ist ein Helligkeits- und Farbsensor, der in drei Modi verwendet werden kann: Reflected Light (aktiv), Ambient Light (inaktiv) und Farbe.

Abbildung 1: EV3 Lichtsensor (LEGO ©)

Befindet sich der Sensor im Ambient Light-Modus, wird das auf den Sensor fallende Umgebungslicht gemessen. Im Reflected Light-Modus leuchtet zusätzlich eine rote LED neben dem eigentlichen Helligkeitssensor, der dann das reflektierte rote Licht einer Fläche misst. Im Farbmodus wechselt der Sensor sehr schnell zwischen einer roten, grünen und blauen LED, die einzeln geschaltet werden und misst die reflektierte Farbintensität, um daraus die Oberflächenfarbe vor dem Sensor berechnen zu können. Da der Sensor nur das Ergebnis der Entscheidung zwischen sieben verschiedenen Farben liefert, ist er nicht als echter Farbsensor, z.B. für den Bau eines Farbscanners geeignet.

Ein Umschalten zwischen den Modi dauert ca. 30 ms. Bei Verwendung des EV3 über USB lässt sich ungefähr alle 9 ms ein Wert auslesen, via Bluetooth, abhängig von Hardware und Rechnerauslastung, sind es nur etwa 10-13 Werte pro Sekunde, was näherungsweise 80-100 ms pro Wert entspricht.

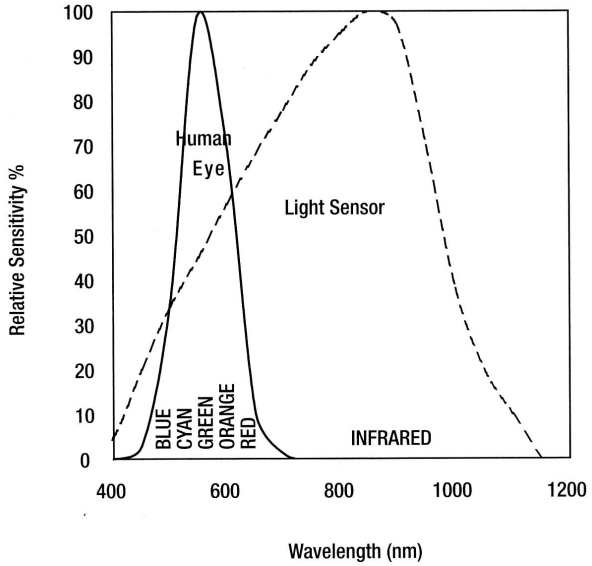


Abbildung 2: Empfindlichkeit des Lichtsensors in Abhängigkeit der Wellenlänge des Lichtes (Michael Gasperi ©)

In Abbildung 2 ist die Empfindlichkeit des Sensors über der Wellenlänge des Lichtes aufgetragen. Zum Vergleich ist auch die Empfindlichkeit des menschlichen Auges abgebildet. Es ist deutlich zu erkennen, dass die maximale Empfindlichkeit des Sensor im infraroten Bereich liegt. Aus diesem Grund ist die LED für den Reflected Light-Modus rot.

Um Werte des Sensors auslesen zu können, muss zunächst eine Bluetooth-Verbindung mit dem EV3-Brick aufgebaut werden, wie Sie es auch schon aus den vorangegangenen Versuchen kennen. Das Objekt für die Kommunikation mit dem EV3-Brick wird im Folgenden als `brickObj` bezeichnet.

`brickObj.sensorX` Sensorobjekt.

`sensorX` Anschluss, an dem der Sensor angeschlossen ist. Mögliche Werte: `sensor1`, `sensor2`, `sensor3`, `sensor4`

```
lightValue = brickObj.sensorX.value;
```

`lightValue` Ausgelesener Wert des Lichtsensors

```
brickObj.sensorX.mode = modeValue;
```

`modeValue` Modus des Sensors. Mögliche Werte: `DeviceMode.Color.Ambient`, `DeviceMode.Color.Reflect`, `DeviceMode.Color.Col`

2.2 Callback-Funktionen

Sowohl bei MATLAB-Timern als auch bei MATLAB-GUIs basiert die Implementierung der durch Sie als Programmierer definierten Funktionalität im Wesentlichen auf Callback-Funktionen. Sie dienen dazu, das sehr allgemeine Gerüst, das z.B. ein Timer-Objekt oder eine GUI bietet, auf einfache und sehr flexible Weise mit der gewünschten eigenen Funktionalität zu füllen.

Um dies zu realisieren, registrieren Sie im Timer-Objekt bzw. in einzelnen GUI-Objekten Funktionen, die bei bestimmten Ereignissen aufgerufen werden sollen. Im Falle des Timers kann beispielsweise in der Eigenschaft `StartFcn` eine Funktion angegeben werden, die ausgeführt wird, sobald der Timer gestartet wird.

Alle Callback-Funktionen haben gemein, dass sie einer Aufrufkonvention folgen müssen. Diese wird von dem Objekt vorgegeben, bei der die Callback-Funktion registriert ist. Wenn es sich um Callback-Funktionen von GUI- oder Timer-Objekten handelt, sind die Rückgabeparameter vorgegeben. In der Regel wird als erster Parameter das Objekt übergeben, welches die Callback-Funktion aufgerufen hat. Weitere Informationen über die Rückgabeparameter entnehmen Sie der MATLAB Dokumentation.

Da bei Callback-Funktionen von Timer- oder GUI-Objekten nicht die Möglichkeit besteht Rückgabeparameter zu definieren, speichern Sie Werte, die Sie später noch benötigen, in einem Attribut der GUI oder in einer Eigenschaft des aufrufenden Objektes. Meist ist dafür – wie beim Timer-Objekt – die Eigenschaft `UserData` vorgesehen. Die darin gespeicherten Daten können später an anderer Stelle im Programm ausgelesen und weiterverarbeitet werden.

Über die Eigenschaft `UserData` des Objektes, welches eine Callback-Funktion aufruft, ist es natürlich nicht nur möglich, Daten aus der Funktion zurückzugeben. Selbstverständlich können darüber der Callback-Funktion auch Daten zur Verfügung gestellt werden. Dies ist wichtig, da das aufrufende Objekt normalerweise immer nur die in der Dokumentation beschriebenen Parameter übergibt. Werden darüber hinaus weitere Daten benötigt, kann dafür kein weiterer Parameter eingeführt werden. Das aufrufende Objekt wüsste nicht, wie es diesen Parameter füllen soll und somit bliebe der Wert undefiniert.

MATLAB erstellt für jede GUI eine Klasse, in der sowohl Methoden, z.B. Callbacks, als auch Attribute angelegt werden können. Dadurch entsteht eine einfache Möglichkeit Daten zu speichern, indem diese in einem Attribut in der GUI abgelegt werden. Weitere Details hierzu erfahren Sie in Kapitel 2.4.5.

2.3 Timer

MATLAB verfügt über einen Mechanismus, der Befehle zu bestimmten Zeitpunkten ausführen kann. Damit können Sie die Ausführung von MATLAB-Code z.B. um eine vorgegebene Zeit verzögern oder eine Funktion in regelmäßigen Abständen wiederholt ausführen lassen. Auch der Start einer Funktion zu einer bestimmten Uhrzeit ist möglich. Im Folgenden wird beschrieben, wie Sie ein Timer-Objekt erzeugen, es starten und wie Sie einstellen, wann der Timer die angegebenen Funktionen ausführt. Um einen Timer zu nutzen gehen Sie wie folgt vor:

1. Erzeugen des Timers
2. Einstellen des Timers (setzen der Werte diverser Parameter und Angabe der MATLAB-Funktionen, die bei Timer-Ereignissen ausgeführt werden sollen)
3. Starten des Timers
4. Löschen des Timers, wenn er nicht mehr gebraucht wird

2.3.1 Erzeugen des Timers

Um einen Timer zu verwenden, müssen Sie zunächst ein Timer-Objekt mit der Funktion `timer` erzeugen. Im folgenden Codebeispiel wird ein einfaches Timer-Objekt erzeugt.

```
>> t = timer
  Timer Object: timer-1

   Timer Settings
      ExecutionMode: singleShot
              Period: 1
           BusyMode: drop
            Running: off

   Callbacks
           TimerFcn: ''
           ErrorFcn: ''
           StartFcn: ''
            StopFcn: ''
```

Einige Einstellungen sind mit Standardeinstellungen vorbelegt. Da keine Callbacks angegeben wurden (im Abschnitt `Callbacks` sind alle Eigenschaften leer), wird beim Starten dieses Timers nichts passieren. Die anderen Einstellungen zeigen, dass der Timer zur Zeit nicht läuft (`Running: off`), eine in `TimerFcn` angegebene Funktion einmal ausgeführt wird (`ExecutionMode: singleShot`) und bei wiederholtem Aufruf der Abstand zwischen zwei Aufrufen eine Sekunde beträgt (`Period: 1`). Die Einstellung `Running` ist nur lesbar. Über sie kann festgestellt werden, ob ein Timer zur Zeit aktiv ist oder nicht.

Die Einstellung `BusyMode: drop` gibt an, dass beim Auslösen des Timers nur dann ein Aufruf einer Callback-Funktion erfolgt, wenn der vorangegangene Aufruf bereits beendet wurde. Löst der Timer hingegen während der Ausführung einer Callback-Funktion aus, wird keine Aktion ausgeführt.

2.3.2 Einstellen des Timers

Die Parameter des Timer-Objektes werden wie in MATLAB üblich über die Funktion `set` gesetzt und mit der Funktion `get` ausgelesen.

```
set(object, 'ParameterName', parameterValue);
parameterValue = get(object, 'ParameterName');
```

Darüber hinaus ist es möglich, die Parameter auch über die Punkt-Notation anzusprechen.

```
object.ParameterName = parameterValue;
parameterValue = object.ParameterName;
```

Wenn Sie die Funktion `set` mit dem Timer-Objekt als Argument aufrufen, können Sie sich die Parameter anschauen, die im Timer-Objekt gesetzt werden können. Falls es Standardwerte gibt, sind diese in geschweiften Klammern angegeben. Alle Felder, inklusive derer, die nur gelesen werden können, sowie deren aktuellen Wert erhalten Sie, wenn Sie die Funktion `get` mit dem Timer-Objekt als Argument aufrufen.

```
>> t = timer;
>> set(t)
    BusyMode: [ {drop} | queue | error ]
    ErrorFcn: string -or- function handle -or- cell array
    ExecutionMode: [ {singleShot} | fixedSpacing | fixedDelay | fixedRate ]
    Name
    ObjectVisibility: [ {on} | off ]
    Period
    StartDelay
    StartFcn: string -or- function handle -or- cell array
    StopFcn: string -or- function handle -or- cell array
    Tag
    TasksToExecute
    TimerFcn: string -or- function handle -or- cell array
    UserData
 >> get(t)
       AveragePeriod: NaN
            BusyMode: 'drop'
            ErrorFcn: ''
       ExecutionMode: 'singleShot'
       InstantPeriod: NaN
                Name: 'timer-1'
    ObjectVisibility: 'on'
```

```
              Period: 1
             Running: 'off'
          StartDelay: 0
            StartFcn: ''
             StopFcn: ''
                 Tag: ''
       TasksExecuted: 0
      TasksToExecute: Inf
             TimerFcn: ''
                 Type: 'timer'
            UserData: []
```

Die wichtigsten Parameter sind:

`Period` Legt den Abstand zwischen den Ausführungen der in `TimerFcn` angegebenen Funktion in Sekunden fest. Werte größer oder gleich 0.001 sind erlaubt.
Standardeinstellung: `1.0`

`ExecutionMode` Über diesen Parameter wird eingestellt werden, ob die in `TimerFcn` angegebene Funktion nur einmal oder mehrfach ausgeführt werden soll und wann die in `Period` angegebene Zeit beginnt abzulaufen. Mögliche Werte sind `'singleShot'` (nur ein Aufruf nach Ablauf der in `StartDelay` angegebenen Zeit), `'fixedRate'` (starte Countdown für nächsten Aufruf, sobald `TimerFcn` in die Ausführungsqueue gestellt wurde), `'fixedDelay'` (starte Countdown für nächsten Aufruf, sobald `TimerFcn` tatsächlich gestartet wurde) und `'fixedSpacing'` (starte Countdown für nächsten Aufruf, sobald `TimerFcn` beendet wurde). Diese sind in der folgenden Abbildung 3 visualisiert.
Standardeinstellung: `'singleShot'`

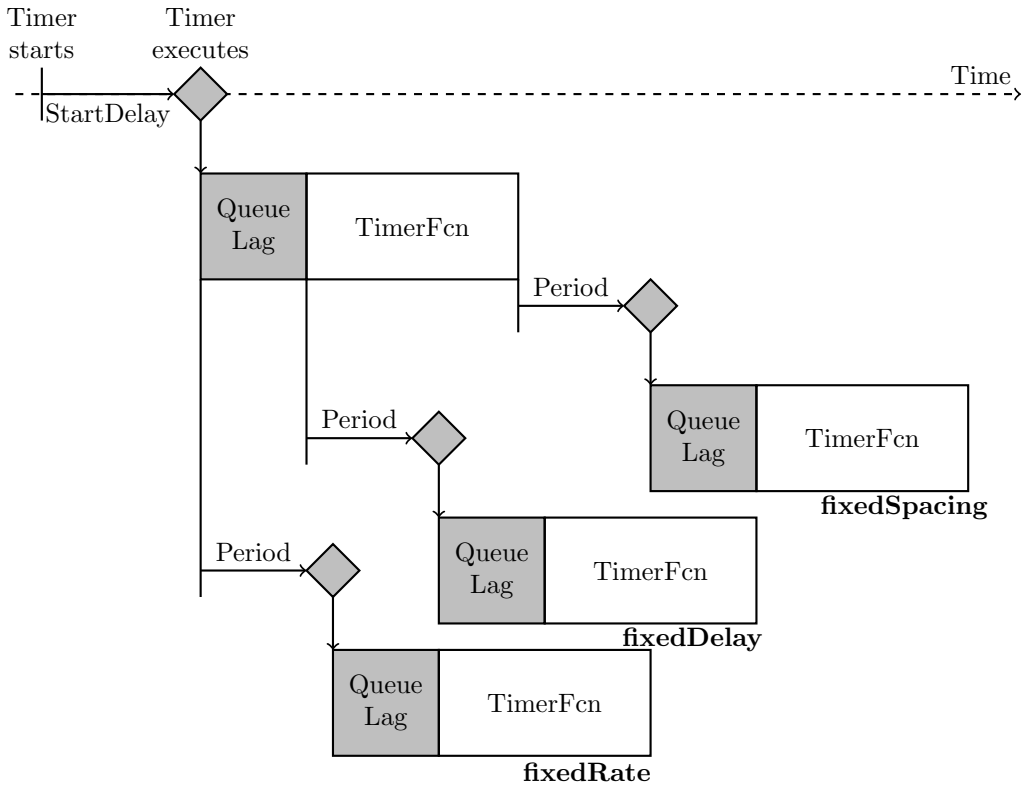


Abbildung 3: Übersicht über die Ausführungsmöglichkeiten eines Timers

`StartFcn, TimerFcn, StopFcn, ErrorFcn` Über diese vier Eigenschaften werden die Callback-Funktionen angegeben, d.h. die Funktionen, die ausgeführt werden sollen, wenn der Timer gestartet bzw. gestoppt wird (`StartFcn` bzw. `StopFcn`). `TimerFcn` ist die Funktion, die einmal bzw. wiederholt aufgerufen wird, während `ErrorFcn` im Fehlerfall ausgeführt wird. Diese Funktionen können auf verschiedene Weisen angegeben werden. Hier soll nur eine Möglichkeiten vorgestellt werden. Die anderen können Sie in der MATLAB-Dokumentation nachlesen.

```
set(t, 'TimerFcn', @callbackFunction);
```

Die Angabe des @-Zeichens vor dem Funktionsnamen ist hierbei zwingend nötig. Damit wird eine Referenz auf die Funktion mit dem auf das @-Zeichen folgenden Funktionsnamen erzeugt. `callbackFunction` bezeichnet dabei eine beliebige Funktion, die folgende Form besitzt

```
function callbackFunction (timerObject, eventStruct)
```

Der Funktionsname kann beliebig gewählt werden. Beim Aufruf der Funktion ist `timerObject` das Timer-Objekt, welches diese Callback-Funktion aufgerufen hat. `eventStruct` ist eine MATLAB-Struktur mit den zwei Feldern `Type` und `Data`. Das Feld `Type` enthält eine Zeichenkette, die den Typ des Ereignisses beschreibt, der

zum Aufruf dieser Funktion führte. Der Wert ist eine der folgenden Zeichenketten: `'StartFcn'`, `'StopFcn'`, `'TimerFcn'` oder `'ErrorFcn'`. Damit ist es möglich, für alle Timer-Ereignisse dieselbe Callback-Funktion zu registrieren und später innerhalb der Funktion festzustellen, welches Ereignis zum Aufruf geführt hat. Für zeitkritische Anwendungen sollte zumindest für `'TimerFcn'` eine eigene Funktion registriert werden, da sonst bei jedem Aufruf das Ereignis über einen zeitintensiven Zeichenkettenvergleich geprüft werden muss.

Das Feld `Data` ist ebenfalls eine Struktur und enthält das Feld `time`, welches die Zeit angibt, zu der das Ereignis auftrat. `time` ist dabei ein Array der Länge 6, wobei die einzelnen Elemente die folgende Bedeutung haben: 1: Jahr, 2: Monat, 3: Tag, 4: Stunde, 5: Minute, 6: Sekunde.

Callback-Funktionen können mit zusätzlichen Parametern verwendet werden. Dies soll hier aber nicht behandelt werden, da es für diesen Versuch nicht relevant ist. Für Details schauen Sie bitte in die MATLAB-Dokumentation.

`UserData` In dieser Eigenschaft können Sie beliebige Daten speichern, die den Callback-Funktionen während ihrer Ausführung zur Verfügung stehen sollen. Wenn Sie die Daten dieser Eigenschaft innerhalb einer Callback-Funktion mit `daten = get(obj, 'UserData')` auslesen und verändern, müssen Sie die Eigenschaft mit `set(obj, 'UserData', daten)` vor Verlassen der Funktion im Timer-Objekt aktualisieren, da die Änderungen sonst verloren gehen.

2.3.3 Starten und Stoppen des Timers

Nachdem das Timer-Objekt den Bedürfnissen entsprechend konfiguriert wurde, muss es gestartet werden. Dafür stehen zwei MATLAB-Befehle zur Verfügung: `start` und `startat`. Diese Funktionen verhalten sich nicht-blockierend (Stichwort: asynchrone Programmierung), d.h. sie kehren sofort zurück und MATLAB fährt im normalen Programmablauf fort. Ist die Eigenschaft `ExecutionMode` auf etwas anderes als `'singleShot'` eingestellt, so muss entweder angegeben werden, wie oft die Funktion in `TimerFcn` maximal aufgerufen werden soll (in der Eigenschaft `TasksToExecute`) oder der Timer muss irgendwann explizit wieder gestoppt werden, damit er nicht immer weiter läuft. Für letzteres ist die Funktion `stop` gedacht. Außerdem kann mit der Funktion `wait` auf die Beendigung eines Timers gewartet werden. Diese Funktion blockiert so lange, bis der Timer nicht mehr läuft. Wenn ein Timer nicht mehr gebraucht wird, sollte das Timer-Objekt mit der Funktion `delete` wieder gelöscht werden, um Speicher freizugeben.

`start` Sofortiges Starten des Timers

```
start(obj)
```

`startat` Starten des Timers zu einer bestimmten Zeit

```
startat(obj, time)
```

`stop` Stoppen des Timers

```
stop(obj)
```

`wait` Warten auf die Beendigung des Timers

```
wait(obj)
```

`delete` Löschen des Timer-Objektes

```
delete(obj)
```

`obj` Das Timer-Objekt, auf dem die Operation ausgeführt werden soll.

`time` Die Zeit, zu der das Timer-Objekt gestartet werden soll. `time` kann dabei in verschiedenen Formaten angegeben werden. Rufen Sie für Details die Dokumentation auf.

2.3.4 Beispiel

Hier nun ein kleines Beispiel. Nehmen wir an, dass folgende Funktion im MATLAB-Pfad verfügbar ist:

```
function myTimerCallback(timerObject, event)

eventTime = event.Data.time;

disp(sprintf('%d-%d-%d %d:%d:%d : %s', ...
    eventTime(1), eventTime(2), eventTime(3), ...
    eventTime(4), eventTime(5), floor(eventTime(6)),...
    event.Type));
end
```

Dann kann der folgende Code in der Konsole eingegeben werden:

```
>> t = timer;
>> set(t, 'ExecutionMode', 'fixedRate');
>> set(t, 'StartFcn', @myTimerCallback);
>> set(t, 'StopFcn', @myTimerCallback);
>> set(t, 'TimerFcn', @myTimerCallback);
>> set(t, 'ErrorFcn', @myTimerCallback);
>> set(t, 'Period', 1);
>> set(t, 'StartDelay', 3);
>> myData.counter = 0;
>> set(t, 'UserData', myData);
>> start(t)
>> pause(10);
>> stop(t);
```

Dies führt dann zu folgender Ausgabe:

```
2008-10-22 12:50:26 : StartFcn
2008-10-22 12:50:29 : TimerFcn
2008-10-22 12:50:30 : TimerFcn
2008-10-22 12:50:31 : TimerFcn
2008-10-22 12:50:32 : TimerFcn
2008-10-22 12:50:33 : TimerFcn
2008-10-22 12:50:34 : TimerFcn
2008-10-22 12:50:35 : TimerFcn
2008-10-22 12:50:36 : StopFcn
```

2.4 Erstellung einer GUI in MATLAB

MATLAB-GUIs sind üblicherweise ereignisgesteuert (engl.: event driven), d.h. die Elemente der GUI reagieren auf Ereignisse, die durch den Benutzer oder andere Prozesse ausgelöst werden. Zu diesen Ereignissen gehören unter anderem Tastatureingaben oder Interaktionen mit der Maus.

Viele von den eingehenden Ereignissen sind für den Programmablauf nicht von Interesse. Für viele Anwendungen ist es beispielsweise nicht relevant, ob sich die Maus gerade hin- und herbewegt oder nicht. Als Programmierer müssen Sie sich nicht mit der enormen Menge an eingehenden Ereignissen beschäftigen, sondern Sie geben MATLAB an, welche für Ihr Programm von Interesse sind und MATLAB kümmert sich dann darum, diese aus den vielen Ereignissen herauszufiltern. Ihr Programm erhält dann die Möglichkeit, darauf zu reagieren.

Damit Ihr Programm auf bestimmte Ereignisse reagiert, müssen Sie Callback-Funktionen definieren und in einem GUI-Element registrieren. Diese werden immer dann ausgeführt, wenn für ein GUI-Element ein entsprechendes Ereignis aufgetreten ist. Für die Zuordnung zwischen GUI-Element, Ereignis und Callback-Funktion wird der Name der Callback-Funktion in die zum Ereignis gehörende Eigenschaft des GUI-Elements eingetragen. Damit weiß MATLAB, welche Funktion aufzurufen ist, wenn das entsprechende Ereignis auftritt. Wenn in der Eigenschaft zu einem Ereignis kein Funktionsname eingetragen ist, so wird dieses Ereignis ignoriert.

Ein GUI-Programm in MATLAB besteht aus zwei Teilen:

1. der Beschreibung der verschiedenen GUI-Elemente, deren Eigenschaften und deren Anordnung innerhalb des Anzeigefensters und

2. dem Quellcode, der in Startfunktion, Ausgabefunktion und Callback-Funktionen unterteilt ist.

Die Trennung zwischen Layout und Funktionalität wird auch in der Oberfläche vom MATLAB App Designer deutlich, den Sie für die Erstellung einer GUI benutzen werden.

In der **Code View** haben Sie die Möglichkeit Callback-Funktionen zu programmieren, während Sie in der **Design View** GUI-Elemente anordnen können.

2.4.1 Verwendung des App Designers

Der MATLAB App Designer ermöglicht es Programmierern in MATLAB komfortabel eine graphische Benutzeroberfläche zu designen und zu programmieren. Die Hauptkomponenten vom App Designer sind die **Design View**, in der Sie der GUI-Elemente per Drag-and-Drop hinzufügen und die **Code View**, in der Sie die Callback-Funktionen der GUI-Elemente programmieren.

Der App Designer startet sich automatisch, wenn Sie unter **Home → New → App** eine neue App anlegen. Alternativ lässt sich der App Designer über den Befehl `appdesigner` starten. Die Abbildung 4 zeigt den App Designer in der **Design View**.

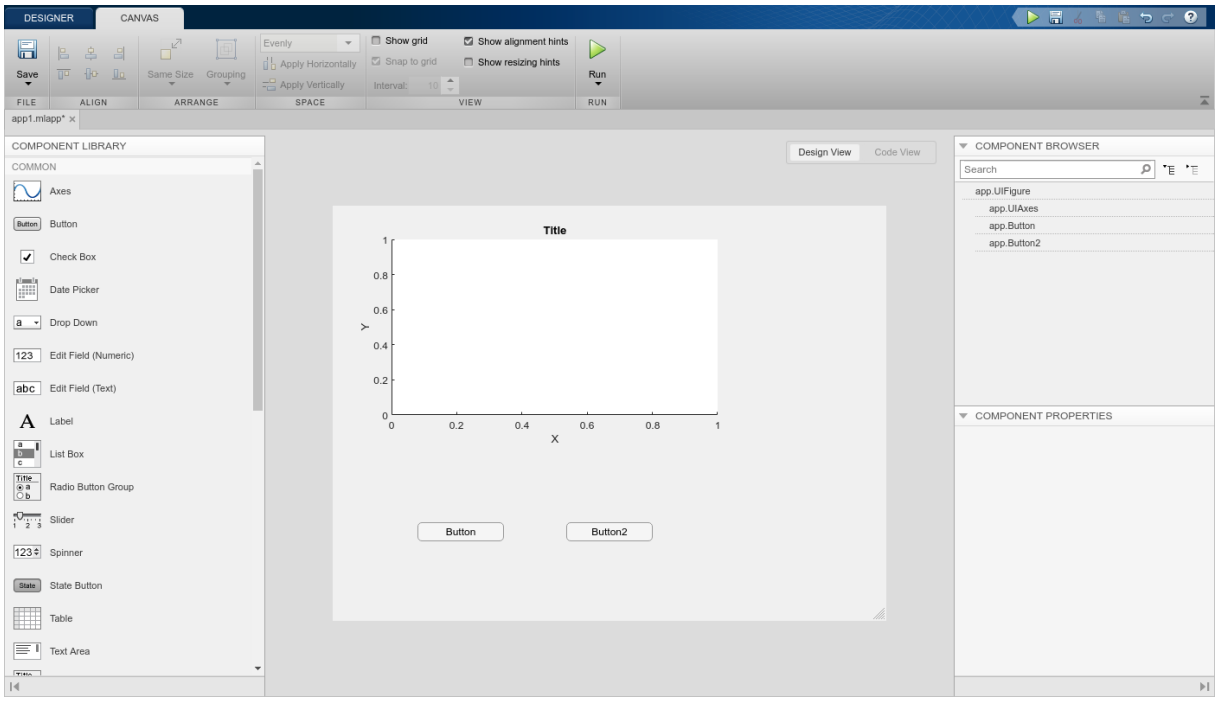


Abbildung 4: Startoberfläche des App Designer

Die neue App wird in einer `.mlapp`-Datei gespeichert, in welcher sich der Code und auch die Anordnung der GUI-Elemente befinden. Zu Beginn erhalten Sie ein leeres Fenster, das von Ihnen mit Elementen befüllt werden kann. MATLAB generiert für diese Elemente in der **Code View** automatisch den benötigten Code für die Initialisierung beim Start der App.

Auch ohne weitere GUI-Elemente ist eine erzeugte App lauffähig. Beim Ausführen der App wird ausschließlich ein leeres Fenster geöffnet, welches keinerlei Funktionalität bietet. Starten können Sie die App wie jede andere Datei von der Konsole über den Dateinamen. Während der Entwicklung können Sie die App auch aus dem App Designer heraus über den Knopf mit dem grünen Dreieck in der Werkzeugleiste starten.

2.4.2 Eigenschaften der GUI-Elemente einstellen

Jedes GUI-Element besitzt eine Reihe von Eigenschaften, die Sie über die **Properties** am rechten Rand einstellen können. Diese öffnen Sie durch einen Klick auf das Element, dessen Eigenschaften Sie verändern wollen. Im **Component Browser** können Sie alle angelegten GUI-Elemente und deren individuelle Bezeichner sehen, die Tags genannt werden. Im Folgenden werden die gebräuchlichsten Eigenschaften `Tag`, `Callback`, `Text` und `Value` näher beschrieben.

`Tag` Das `Tag` dient als eindeutiger Identifikator eines Elements innerhalb einer GUI. Der App Designer weist jedem Element einen eindeutigen Namen zu, den Sie über den **Component Browser** mit einem Rechtsklick in einen möglichst aussagekräftigen Namen ändern sollten. Dieser Name muss ein gültiger MATLAB-Variablenname sein. Der App Designer nutzt diese Eigenschaft um

- den Namen generierter Callback-Funktionen zu konstruieren,
- die korrespondierende Callback-Eigenschaft so zu setzen, dass sie auf die Callback-Funktion verweist und
- ein Feld mit diesem Namen in der `app` – die Sie später noch kennen lernen werden – anzulegen, welches ein Handle auf das GUI-Objekt enthält.

`Callback` Die Eigenschaft `Callback` spezifiziert die Funktion, die bei Aktivierung des Elements ausgeführt werden soll. Diese Eigenschaft wird typischerweise bei Knöpfen benutzt, um auf einen Klick auf den Knopf zu reagieren. `Callback` ist aber nur eine von vielen Eigenschaften, die Verweise auf Callback-Funktionen enthalten können. Eine Übersicht aller Eigenschaften mit denen Sie ebenfalls Callback-Funktionen spezifizieren können, für welche GUI-Elemente sie zur Verfügung stehen und bei welchen Ereignissen die Callback-Funkionen dann ausgeführt werden, finden Sie in der Dokumentation, wenn Sie nach *Overview* und *Callbacks* suchen. Die genaue Syntax ist außerdem unter *Callback Definition* in der MATLAB-Dokumentation erläutert.

`Text` Die Eigenschaft `Text` beinhaltet den Text eines GUI-Elements. Bei Knöpfen, Checkboxen und statischen Textfeldern ist dies der Text, der auf oder neben den entsprechenden GUI-Elementen angezeigt wird. Je nach Typ können Sie hier auch mehrzeiligen Text eingeben.

`Value` Die Eigenschaft `Value` beinhaltet einen numerischen Wert für ein GUI-Element, welcher in dem Bereich liegen muss, der durch die Eigenschaften `Min` und `Max` begrenzt wird. Diese Eigenschaft werden Sie häufig bei Radio-Knöpfen, Auswahlboxen

und Schiebereglern verwenden, um den aktuellen Status bzw. die aktuelle Position zu setzen oder während der Programmausführung auszulesen.

Die Bedeutung der anderen Eigenschaften und welche Eigenschaften für die einzelnen Elemente spezifisch sind, entnehmen Sie bitte der MATLAB-Dokumentation. Am schnellsten werden Sie fündig, wenn Sie nach dem Element in Kombination mit *properties* suchen.

2.4.3 Aufbau der Code View

Die **Code View** ist der zentrale Bestandteil der MATLAB-Anwendung. Während die **Design View** nur die visuelle Darstellung einiger Teile des Codes umfasst und Sie im Aufbau der App unterstützen soll, beinhaltet die **Code View** den gesamten Code der GUI.

Die App besteht aus einer Klasse, die sich in Methoden (`methods`) und Attribute (`properties`) unterteilt. Teile des Codes sind dabei von MATLAB generiert und ausgegraut, sodass Sie diese nicht oder nur über die **Design View** verändern können. Methoden und Attribute können als `public` oder `private` definiert werden, wobei als `private` definierte Methoden und Attribute nur innerhalb der App benutzbar sind.

Aufbau der app Über die `app` kann mit der Punkt-Notation auf alle Funktionen und Attribute zugegriffen werden, die innerhalb der App angelegt wurden.

```
properties (Access = public)           (1)
    text1                              (2)
end

methods (Access = private)             (3)

    % function test for example
    function setText1()                (4)
        app.text1 = 'Hello World!';    (5)
    end
end
```

Im oberen Codebeispiel sind ein Attribut (2) und eine Funktion (4) angelegt worden. Die Attribute können im Bereich der `properties` (1) deklariert werden, während die Funktionen sich im Bereich `methods` (3) befinden.

In Zeile 1 sind Attribute angegeben, deren Zugriff `public` ist. Auf das Attribut `text1` kann mittels `app.text1` zugegriffen werden (5). Die Methoden der App sind `private` (3). Die Methode `setText1` setzt den Wert des Attributes `text1` auf `Hello World!` (5). Auf die Methode kann mittels `app.setText1()` zugegriffen werden.

In Callback-Funktionen wird die `app` immer als erstes Argument übergeben, sodass Daten innerhalb der app genutzt werden können.

Starten der GUI (StartupFcn) Diese Funktion wird direkt nach dem Starten der GUI, aber noch bevor Sie dem Benutzer angezeigt wird, aufgerufen. Als Parameter wird die `app` übergeben. Neben dem Standardparameter können der Funktion Kommandozeilenparameter übergeben werden, die über den Button **App Input Arguments** in der **Code View** angelegt werden. Die Funktion wird zum Beispiel benutzt um Standardwerte zu setzen, Variablen zu initialisieren oder Befehle auszuführen, welche die Initialisierung der GUI betreffen.

Über einen Rechtsklick auf den Hintergrund der App in der **Design View** kann ein Callback für die StartupFcn generiert werden.

Schließen der GUI (CloseRequestFcn) Diese Funktion wird beim Schließen der App aufgerufen und wird unter anderem dazu verwendet laufende Timer zu beenden oder Objekte zu löschen.

Über einen Rechtsklick auf den Hintergrund der App in der **Design View** kann ein Callback für die CloseRequestFcn generiert werden.

Callback-Funktionen Wenn ein Benutzer ein GUI-Element aktiviert, führt die GUI, falls für dieses Ereignis eine Callback-Funktion registriert wurde, die entsprechende Funktion aus. Der Name der Callback-Funktion wird durch die Eigenschaft `Tag` und dem zu behandelnden Ereignis bestimmt. Zum Beispiel wird bei Aktivierung eines Push-Buttons, dessen `Tag`-Eigenschaft auf `printButton` gesetzt ist, die folgende Callback-Funktion aufgerufen:

```
% Button pushed function: printButton
function printButtonPushed(app, event)
  % Code ...
end
```

Beim Anlegen eines neuen GUI-Elements wird automatisch durch den App Designer ein Eintrag in der `createComponents`-Methode angelegt, welche die App erstellt und initialisiert. Diese Aufgabe übernimmt MATLAB für Sie. Für Callback-Funktionen wird ein Grundgerüst generiert, sobald durch einen Rechtsklick auf das jeweilige GUI-Element und **Callbacks➔add ... callback** in der **Design View** geklickt wird.

2.4.4 Callback-Funktionen

Eine MATLAB-GUI-Anwendung ohne Callback-Funktionen ist nicht funktional. Wenn Sie alle GUI-Elemente nach Ihren Wünschen angeordnet haben, müssen diese mit Interaktionen versehen werden. Dies machen Sie, indem Sie für die gewünschten Ereignisse Callback-Funktionen registrieren und dann diese Funktionen implementieren.

In der **Design View** finden Sie in den **Properties** einen Reiter `Callbacks`. Dieser listet alle für dieses GUI-Element verfügbaren Callback-Typen auf. Über das Dropdown-Menü können Sie dem Callback-Typ eine Callback-Funktion zuweisen. Sofern Sie eine Funktion

mithilfe des App Designers generiert haben, ist diese standardmäßig ausgewählt. Der Code, der durch den App Designer bei neuen Funktionen generiert wird, sieht typischerweise wie folgt aus:

```
% Value changed function: CheckBox
function CheckBoxValueChanged(app, event)
  value = app.CheckBox.Value;
  % Code ...
end
```

Über den Parameter `app` haben Sie Zugriff auf alle Objekte, Attribute und Methoden, die in ihrer App zur Verfügung stehen. Darunter auch das Objekt, das die Callback-Funktion ausgelöst hat (siehe Codebeispiel oben). Mithilfe der Funktionen `get` und `set` haben Sie darüber Zugriff auf dessen Eigenschaften. Im Folgenden sehen Sie ein paar Beispiele, um die Möglichkeiten zum Auslesen und setzen der Eigenschaften von GUI-Elementen innerhalb einer Callback-Funktion zu verdeutlichen.

```
% Status einer Auswahlbox 'Checkbox' abfragen bzw. setzen
status = get(app.CheckBox,'Value');
status = app.CheckBox.Value;          % identisch
set(app.CheckBox,'Value', 0);
app.CheckBox.Value = 0;               % identisch

% Beschriftung eines Knopfes 'Button' ändern
set(app.Button, 'Text', 'Changed');
```

Weitere Informationen zu den Callback-Funktionen erhalten Sie in der MATLAB-Dokumentation in den Properties der verwendeten GUI-Elemente.

2.4.5 Datenverwaltung

Der zentrale Dreh- und Angelpunkt für die Verwaltung von Daten, die innerhalb der GUI gespeichert und zwischen den Callback-Funktionen weitergereicht werden, ist die `app`. Sie können mithilfe der `properties` eigene Strukturen innerhalb der `app` anlegen, die jeder Callback-Funktion übergeben wird. Dadurch können Sie sowohl auf alle Funktionen innerhalb der `app` zugreifen, indem Sie den Punktoperator verwenden, als auch auf die Daten, die Sie in den `properties` abgelegt haben.

Der `app` können Sie nun beliebige Daten hinzufügen, die dann allen Callback-Funktionen zur Verfügung stehen. Hier ist es sinnvoll, Ihre Daten deutlich von den automatisch angelegten Elementen zu trennen, indem Sie Unterstrukturen anlegen. Konfigurationsdaten Ihrer GUI könnten Sie z.B. in `app.config.*` ablegen und andere zwischengespeicherte Daten in `app.data.*`. Zu den automatisch angelegten Attributen zählen zum Beispiel die Einträge der GUI-Elemente.

Je nach Funktionalität Ihrer GUI kann es vorkommen, dass sich Daten in der `app` während der Ausführung einer Funktion verändert haben. Dies ist z.B. dann der Fall, wenn Sie in einer Funktion längere Berechnungen vornehmen und währenddessen Benutzereingaben zulassen, durch die bestimmte Daten verändert werden. Damit diese Veränderungen keinen Einfluss auf ihre aktuelle Funktion haben, sollten Sie vorher eine lokale Kopie der Daten in ihrer Funktion erstellen. Vergessen Sie nicht, diese Daten nach der Verwendung in der `app` zu speichern und seien Sie vorsichtig, dass Sie keine Daten überschreiben!

Beispiel

Sie können die `app` nutzen, um eine Abbruchfunktion zu implementieren. Dabei setzt die Callback-Funktion des Abbrechen-Knopfes eine Statusvariable in der `app`. Diese wird in der zeitintensiven Funktion periodisch überprüft. Springt die Statusvariable auf einen bestimmten Wert, beendet sich die Funktion.

3 Durchführung

3.1 Sensor auslesen (90 Min)

Für die Durchführung der folgenden Aufgaben stehen Grundgerüste zweier kleiner MATLAB-Funktionen zur Verfügung, mit denen Sie die Verbindung von MATLAB zum EV3 herstellen und den Sensor in den passenden Modus setzen bzw. die Verbindung schließen. Der Farbsensor wird in diesem Versuch nur in den Modi *ambient* und *reflect* an Sensorport 1 verwendet. Die Funktionen sind wie folgt definiert:

`lightConnectEV3` Verbindung zwischen MATLAB und dem EV3 herstellen und den Lichtsensor initialisieren.

```
brickObj = lightConnectEV3(brickConnectionType, sensorMode)
```

`brickConnectionType` String, der den Verbindungstyp angibt. Mögliche Werte: `'usb'`, `'bluetooth'`

`sensorMode` String, der den Modus angibt, in dem der Sensor aktiviert werden soll. Mögliche Werte: `'ambient'`, `'reflect'`

`lightDisconnectEV3` Verbindung zwischen MATLAB und dem EV3 trennen

```
lightDisconnectEV3(brickObj)
```

Die Gerüste für die Funktionen finden Sie in den Dateien `lightConnectEV3.m` und `lightDisconnectEV3.m`. Die Stellen, an denen Sie Code einfügen sollen, sind mit `IHR CODE HIER` gekennzeichnet.

a) Ergänzen Sie das vorgegebene Gerüst in der Funktion `lightConnectEV3`, so dass der Lichtsensor abhängig von dem übergebenen Parameter entweder in den *ambient*- oder in den *reflect*-Modus gesetzt wird.

Ergänzen Sie danach das Gerüst in der Funktion `lightDisconnectEV3`, so dass die Verbindung zum EV3 getrennt wird.

Testen Sie nun die beiden Funktionen mit verschiedenen Parametern.

b) Lesen Sie als nächstes bei verbundenem EV3 und dem Sensor im ambient-Modus mehrere Helligkeitswerte aus, während Sie z.B. mit der Hand den Lichteinfall auf den Sensor variieren.

Welche Werte können Sie maximal bzw. minimal erreichen?

c) Lesen Sie nun Helligkeitswerte im aktiven Modus aus, während Sie den Sensor über verschieden dunkle Flächen halten.

Welche Werte können Sie maximal bzw. minimal erreichen? Wie stark unterscheiden sich die Helligkeitswerte der verschiedenen Farben?

d) Schreiben Sie nun eine Funktion `lightReadWithLoop` in der Sie eine durch den Parameter `numberOfSeconds` vorgegebene Anzahl von Sekunden lang Sensorwerte nacheinander einlesen (z.B. in einer `while`-Schleife). Verwenden Sie den Rahmen aus der Datei `lightReadWithLoop.m`. Beachten Sie die gegebenen Funktionsargumente. Nach dem Einlesen der Sensorwerte sollen diese in einen Graphen geplottet werden.

Zum Testen bauen Sie zunächst mit `lightConnectEV3` eine Verbindung zum EV3 auf und führen dann `lightReadWithLoop` aus, um die Messung durchzuführen. Bewegen Sie den Sensor während der Messung über verschiedenfarbige Flächen (im aktiven Modus) bzw. aus dem Schatten zu einer Lichtquelle (im inaktiven Modus).

HINWEIS: Mit den Funktionen `tic` und `toc` lässt sich in MATLAB eine einfache Zeitmessung durchführen.

e) Erweitern Sie die Funktion `lightReadWithLoop` so, dass zu jedem Sensorwert die aktuelle Zeit gespeichert wird.

Geben Sie anschließend die Sensorwerte über der Zeit in einem Graphen aus und achten Sie auf sinnvolle Achsenbeschriftungen. Dabei soll die Zeitachse die Sekunden ab dem ersten Sensorwert darstellen, d.h. die Zeit des ersten Sensorwertes ist Null Sekunden.

Plotten Sie weiterhin in ein eigenes Fenster die Zeitdifferenzen zwischen aufeinander folgenden Messungen über der Zeit und bestimmen Sie den Mittelwert dieser Zeitdifferenzen. Plotten Sie diesen Mittelwert als waagerechte Linie zusätzlich in das gleiche Fenster.

f) Bestimmen Sie für die verschiedenen Kommunikationswege (Bluetooth und USB), welche Zeit im Mittel zwischen zwei Messungen liegt. Gibt es deutliche Ausreißer? Welche Unterschiede beobachten Sie zwischen einer Bluetooth- und einer USB-Verbindung?

3.2 Timersteuerung (120 Min)

In dieser Aufgabe ist das Einlesen der Sensorwerte in einer Schleife (`lightReadWithLoop`) durch einen Timer zu ersetzen, der periodisch das Einlesen eines Sensorwertes anstößt. Verwenden Sie als Rahmen die Datei `lightReadWithTimer.m`. Im Folgenden sollen Sie Ihre Lösung aus `lightReadWithLoop` schrittweise umbauen und dann in den neuen Rahmen überführen, um zur Lösung mit Timer zu kommen.

a) Verlegen Sie die Ergebnisvektoren (Sensorwerte und Messzeitpunkte) sowie das EV3-Objekt in der Funktion `lightReadWithLoop` in eine Struktur-Variable `myUserData`.

Schreiben Sie den Rest der Funktion so um, dass Sie ausschließlich auf Daten in der Struktur `myUserData` arbeiten. Dafür müssen Sie ggf. weitere Felder anlegen. Anschließend testen Sie die Funktion.

b) Kopieren Sie die Initialisierung der Struktur-Variable `myUserData`, den Start der Zeitmessung und die graphische Ausgabe in die Funktion `lightReadWithTimer`. Erzeugen Sie nach der Initialisierung von `myUserData` ein Timer-Objekt und initialisieren Sie die Eigenschaften `Period`, `UserData`, `TimerFcn` und `ExecutionMode`. Der Timer soll alle 100 ms (bei Bluetooth-Verbindung) bzw. 50 ms (bei USB-Verbindung) die Funktion `readLightTimerFcn` aufrufen. Den Callback-Funktionen sollen die Werte aus `myUserData` zur Verfügung gestellt werden.

c) Im nächsten Schritt verlagern Sie den Code aus der Schleife in `lightReadWithLoop`, in der Sie die Sensorwerte und die Zeiten abspeichern, in die Funktion `readLightTimerFcn`. Beachten Sie, dass Sie nun zusätzlich als erstes die Daten für `myUserData` aus dem Timer-Objekt holen und als letztes die geänderten Daten darin wieder abspeichern müssen.

d) Nun soll der Timer verwendet werden, um die Funktion `lightReadTimerFcn` periodisch zu starten. Starten Sie dazu nach dem Start der Zeitmessung den Timer. Überlegen Sie sich einen Mechanismus, um den Timer nach der angegebenen Anzahl von Sekunden wieder zu stoppen. Welche Möglichkeiten gibt es hier? Implementieren Sie eine Möglichkeit und testen Sie Ihre nun fertige Funktion.

   HINWEIS: Die Zeitmessung kann weiterhin mittels `tic` und `toc` erfolgen..

e) Variieren Sie die Einstellung der Eigenschaft `ExecutionMode` des Timers. Welches Verhalten können Sie bei den Abständen der Messungen beobachten? Wie ist das Verhalten zu erklären?

3.3 GUI Programmierung (120 Min)

Nun sollen Sie eine kleine GUI programmieren, mit der Sie das Einlesen neuer Sensordaten mit einem Knopfdruck starten bzw. stoppen und den Sensor-Modus umschalten können. Während der Erfassung der Daten sollen diese in dem GUI-Fenster angezeigt und ständig aktualisiert werden.

a) Öffnen Sie mit dem Befehl

```
>> appdesigner lightGui
```

   die vorbereitete GUI. Öffnen Sie den Quelltext der GUI über die **Code View** und schauen Sie sich die Funktion `startupFcn` an. Diese Funktion wird kurz vor Anzeige der GUI ausgeführt. Damit können Sie Voreinstellungen und Initialisierungen vornehmen.

   In unserem Fall wird hier die Funktion `lightConnectEV3` mit entsprechenden Parametern aufgerufen, um die Verbindung zum EV3 aufzubauen und den Lichtsensor zu aktivieren. Außerdem werden einige Konfigurationsparameter gesetzt, sowie das Timer-Objekt erzeugt und konfiguriert.

Passen Sie in dieser Funktion die entsprechenden Stellen an, so dass der gewünschte Verbindungstyp (USB oder Bluetooth) und der richtige Modus für den Lichtsensor genutzt werden.

Am Anfang der Datei sind Gerüste für die Timer-Callback-Funktionen angelegt. Die restliche Datei ist vom App Designer automatisch generiert worden. Im Folgenden werden Sie weiteren Code erzeugen, um die GUI zum Leben zu erwecken.

b) Beim Schließen der GUI soll alles ordentlich aufgeräumt werden. Erzeugen Sie deshalb eine Callback-Funktion für das Ereignis `CloseRequest` der GUI, indem Sie in der **Design View** auf den Hintergrund mit der rechten Maustaste klicken und **Callbacks➔Go to UIFigureCloseRequestFcn callback** auswählen. Alternativ können Sie den Callback auch über einen Rechtsklick im Component Browser auf `app.UIFigure` auswählen.

In der erzeugten Funktion stoppen und löschen Sie den Timer, deaktivieren den Sensor und schließen die Verbindung zum EV3. Zuletzt löschen Sie das Figure-Objekt (mit `delete`). Der Code für Letzteres wurde vom App Designer bereits automatisch generiert.

Theoretisch könnten Sie all dies auch in der Callback-Funktion des Quit-Knopfes tun. Bei Beenden der GUI auf eine andere Weise als den Quit-Knopf wird die Callback-Funktion des Quit-Knopfes allerdings niemals aufgerufen und folglich wird weder der Timer gestoppt noch die Bluetooth-Verbindung getrennt. Die Callback-Funktion für `CloseRequestFcn` wird hingegen bei allen Versuchen, die GUI zu beenden, aufgerufen, so dass in jedem Fall alles beendet wird.

Falls das Schließen der GUI mal nicht möglich sein sollte (z.B. wenn in dieser Callback-Funktion ein Fehler auftritt) können Sie die GUI dennoch von der MATLAB-Konsole durch Eingeben von

```
>> close force
```

schließen.

c) Erzeugen Sie nun für den Quit-Knopf die Callback-Funktion. Klicken Sie dafür mit der rechten Maustaste auf den Quit-Knopf und wählen Sie aus dem Popup-Menü **Callbacks➔Go to QuitButtonPushed callback**. Damit gelangen Sie zum von MATLAB generierten Funktionsrumpf der Funktion, die aufgerufen wird, wenn dieser Knopf angeklickt wird. Veranlassen Sie darin das Schließen der GUI mit

```
close(app.UIFigure);
```

Dies führt zu einem `CloseRequest`, für den Sie im vorangegangenen Aufgabenpunkt die Callback-Funktion geschrieben haben.

Testen Sie, ob sich die GUI öffnen und ohne Fehlermeldungen auch wieder schließen lässt. Testen Sie dabei insbesondere, ob beim Schließen der GUI über den X-Knopf rechts oben im Fenster Fehler auftreten.

d) Belegen Sie den Start-Knopf mit seiner Funktion. Schreiben Sie dafür den Code zum Starten des Timers, der in `app.myTimer` gespeichert ist, in die entsprechende Callback-Funktion. Sorgen Sie dafür, dass der Timer nur gestartet wird, wenn er nicht bereits läuft, da es sonst eine Fehlermeldung gibt. Zum Vergleich von Zeichenketten verwenden Sie die Funktion `strcmp`.

   Starten Sie die GUI und testen Sie den Start-Knopf. Wenn der Timer erfolgreich startet, erscheint eine Meldung auf der Konsole darüber, da die Eigenschaft `StartFcn` des Timers entsprechend konfiguriert ist.

e) Gehen Sie für den Stop-Knopf ebenso vor, nur dass Sie hier den Timer stoppen.

   Starten Sie wiederum die GUI und testen Sie den Stop-Knopf. Wenn der Timer erfolgreich stoppt, erscheint eine Meldung darüber, da die Eigenschaft `StopFcn` des Timers entsprechend konfiguriert ist.

f) Gehen Sie nun zur Callback-Funktion des „Toggle Mode“-Knopfes. In dieser Funktion stoppen Sie zunächst den Timer, falls dieser läuft. Dann setzen Sie den Wert von `app.config.sensorMode` von `'ambient'` auf `'reflect'` bzw. umgekehrt. Anschließend aktualisieren Sie den Auslesemodus am Lichtsensor. Falls Sie den Timer gestoppt haben, starten Sie den Timer.

   Aktualisieren Sie am Ende der Funktion schließlich die Modusanzeige in der GUI mit:

```
app.ModeText.Text = app.config.sensorMode;
```

   Starten Sie die GUI und testen Sie das Zusammenspiel aller Knöpfe. Es müssen wiederum entsprechende Meldungen des Timers erscheinen. Anhand der LED des Lichtsensors können Sie den aktuellen Modus überprüfen.

g) Schließlich fehlt noch der Inhalt der Timer-Callback-Funktion. Hier werden, wie vorher auch, die Sensorwerte abgespeichert. Die Menge der Sensordaten soll allerdings auf `app.config.numberOfValues` Werte begrenzt werden. Außerdem sollen die Daten geplottet werden. Die Variable für die Sensordaten (`myUserData.measurementData`) wird für Sie schon in der Funktion `startupFcn` mit einer festen Anzahl Nullen initialisiert.

   Nachdem Sie die Daten aus dem Timer-Objekt geholt haben, verschieben Sie nun als erstes alle Messwerte in der entsprechenden Variable um einen Wert zu kleinen Indizes. Setzen Sie dann den letzten Wert des Vektors auf den neuen Sensorwert. Zum Schluss plotten Sie die Daten in das Axes-Element `app.UIAxes`.

   Am Ende der Callback-Funktion speichern Sie die geänderten Daten im Timer-Objekt.

   Wenn Sie nun die GUI laufen lassen und den Timer starten, sollten ständig neue Werte vom Sensor gelesen werden und die Anzeige entsprechend aktualisiert werden.

Machen Sie nun die folgenden Experimente mit dem Sensor im aktiven und passiven Modus:

- Überprüfen Sie, bei welchem Abstand zu einer weißen Fläche im aktiven Modus die größte Helligkeit detektiert wird.
- Scannen Sie verschiedene Farben in der LEGO-Bauanleitung des Roboters. Welche Kombination dieser Farben würden sich für eine Farbunterscheidung eignen, welche nicht?
- Wenn Sie noch Zeit haben, legen Sie für die Farben, die sich gut für eine Farbunterscheidung eignen, Schwellwerte fest und fügen Sie der GUI die Funktionalität eines einfachen Farberkenners hinzu.