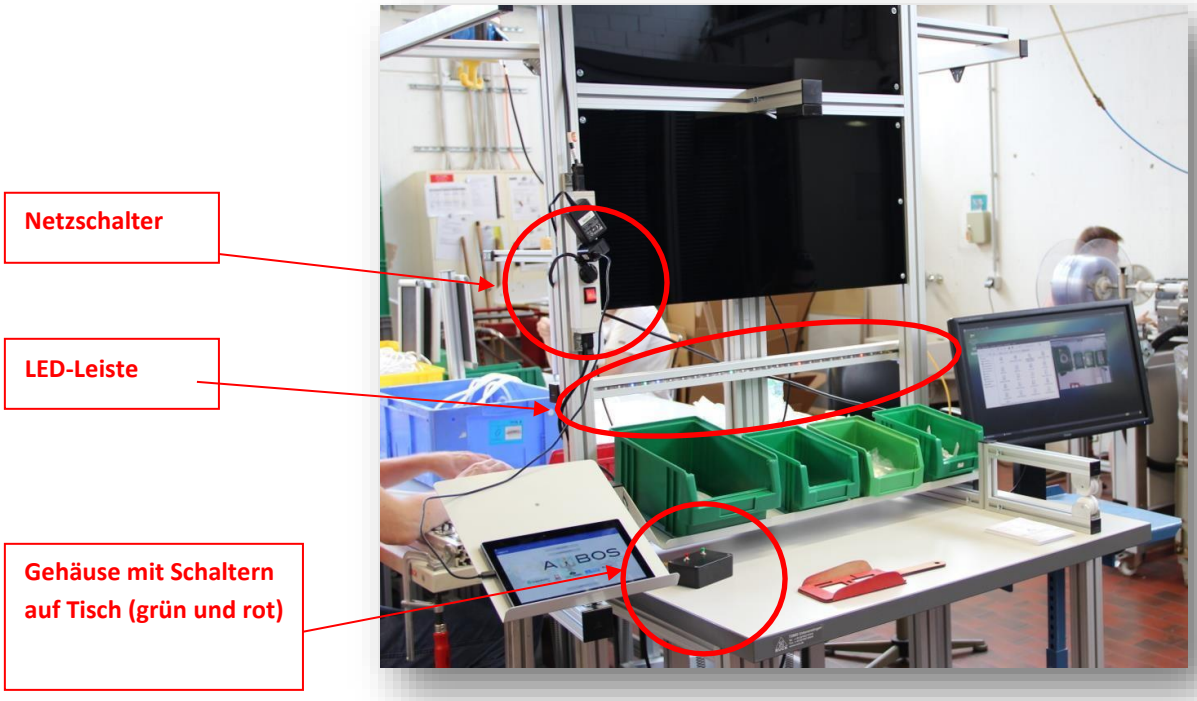
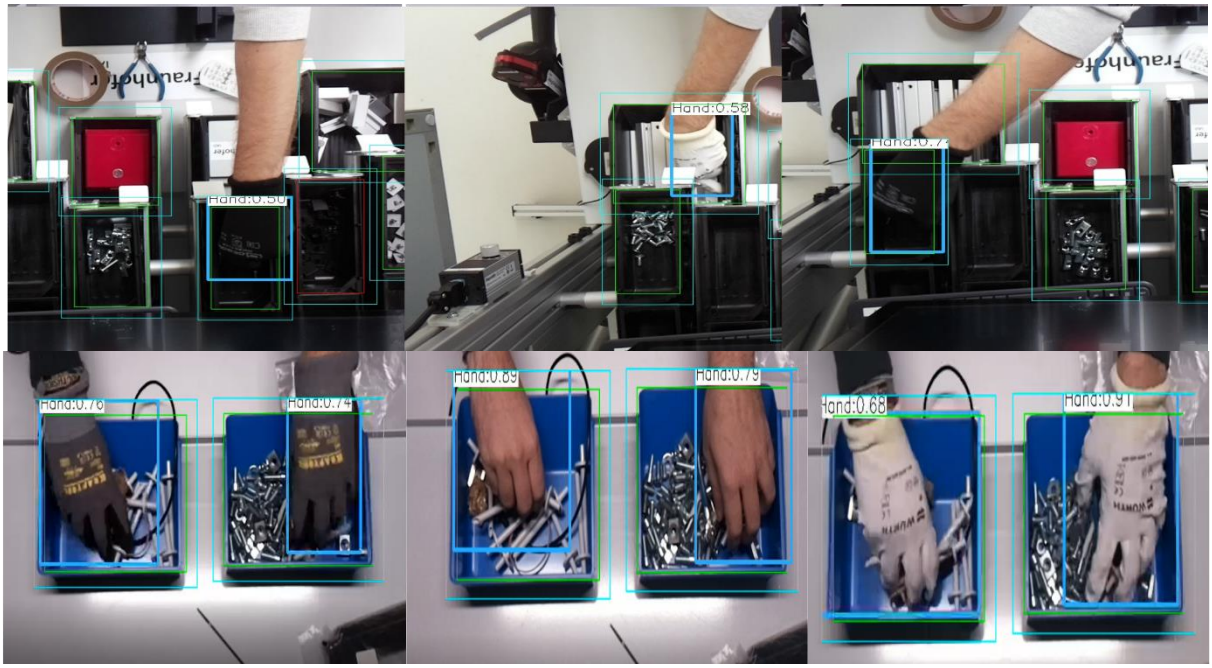

AMBOS
-3D
kontrolliert
Fraunhofer
IPA
caritas
STUTTGART
HOBBYHIMMEL
DIE OFFENE WERKSTATT
RUCK
AluConnect
Bundesministerium
für Bildung
und Forschung


Netzschalter
LED-Leiste
Gehäuse mit Schaltern
auf Tisch (grün und rot)


Hand:0.58
Hand:0.76
Hand:0.74
Hand:0.89
Hand:0.79
Hand:0.68
Hand:0.91

Kurzbeschreibung

Eine allgemeine Kurzbeschreibung des Ambos-Systems ist auf der Homepage zu finden.

Zusätzliche Information

Zentrale Komponente von Ambos ist die Erkennung einer Entnahme eines Bauteils aus einer Box (jeglicher Form, Farbe, …). Im ursprünglichen System wurde jede sich bewegende Kontur im Bild als mögliche Hand identifiziert. Dieser Ansatz war recheneffizient, führte aber zu Problemen bei variierenden Lichtverhältnissen. Das System wurde überarbeitet und auf einen leistungsstärkeren Mikrocontroller übertragen. Dadurch konnte eine ML basierte Handerkennung (Yolov4) integriert und die Robustheit des Systems verbessert werden. Der „alte“ Ansatz kann jedoch immer noch verwendet werden, indem man die ML-Handerkennung manuell deaktiviert wird.

Momentan wird an zwei Versionen der Ambos Applikation gearbeitet. Einer *standalone* – Variante (wie bisher), bei welcher die komplette Prozesslogik (z.B. das Mitzählen von entnommenen Bauteilen) von Ambos selbst übernommen wird und eine integrierte Version (v2), bei welcher die Steuerung des Prozesses über eine *Web* – und *ControlApp* erfolgt (hier meldet Ambos, aka die *SensorApp*, lediglich erkannte Entnahmen – ob diese richtig waren und was dies für den Packprozess bedeutet wird von der *ControlApp* entschieden).

Das System im Betrieb

Da Version 2 der Ambos-Applikation noch **in Bearbeitung** ist, wird standardmäßig die **standalone** Variante gestartet. Für Testzwecke kann aber bereits auf die in Web-/ControlApp integrierte Lösung umgestellt und der Prozess über exemplarische MQTT Nachrichten simuliert werden. Mehr Informationen weiter unten.

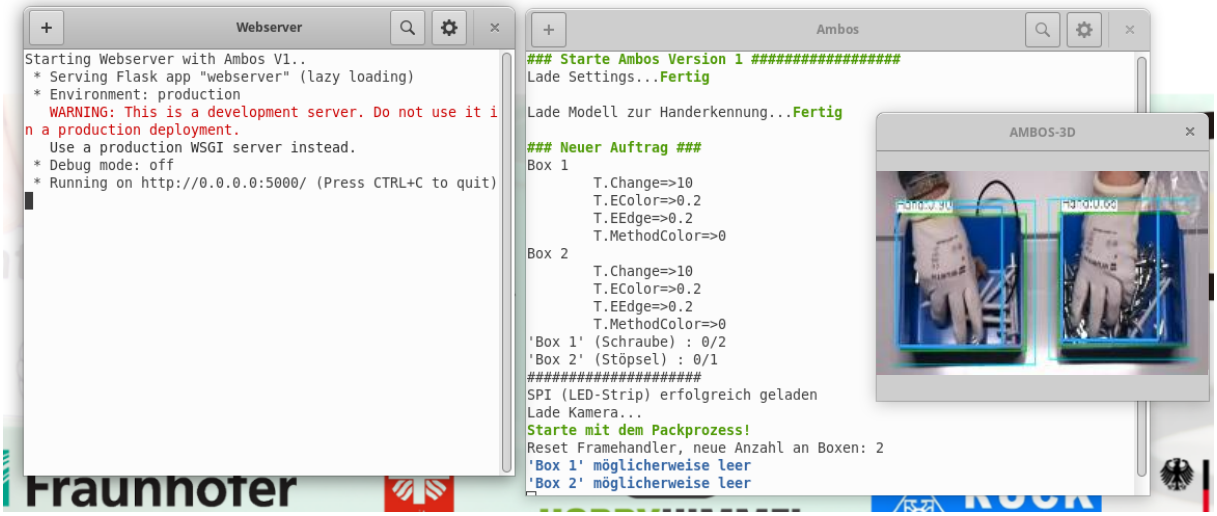
Sobald das Image auf einer SD-Karte vorliegt und alle notwendigen Hardware-Komponenten (LED, Kamera, Taster, evtl. Tastatur/Maus/Bildschirm) am Pi angeschlossen sind, kann die Karte in den dafür vorgesehenen Slot gesteckt und das System gestartet werden.

Start

Das System kann entweder mit oder ohne angeschlossenen Monitor gestartet werden. Die Verwendung eines Monitors empfiehlt sich, wenn man den Prozess visualisieren und zusätzliche Informationen erhalten möchte

Netzschalter umlegen: Zum Starten wird lediglich der Strom angeschaltet, das gesamte System fährt dann automatisch hoch. Die LEDs beginnen zu leuchten. Das System ist einsatzbereit, sobald die LED-Startsequenz vorüber ist und die LEDs über den Boxen alle rot leuchten. (Achtung: Es ist möglich, dass dazu noch einmal der grüne Schalter auf dem Tisch gedrückt werden muss („Neustart der Erkennung“))

Zusätzlich zur eigentlichen Ambos Applikation wird ein kleiner Webserver mitgestartet. Über diesen Webserver kann ein Auftrag erstellt werden und das aktuelle Frame abgefragt werden, um Boxen einzuzeichnen. Dies kann aber auch alles direkt über manuelle Änderungen in den entsprechenden Settings Dateien erfolgen. Ist ein Monitor angeschlossen, so trifft man nach dem Boot auf folgende Fenster: Das linke Terminal zeigt den Webserver, der automatisch die Ambos Applikation startet.



Hierfür wird eine Order-Datei und Settings-Dateien ausgelesen (siehe unten) und die Handerkennungen (falls aktiviert) gestartet. In der Visualisierung werden die eingezeichneten Boxen (+Padding) und erkannte Hände angezeigt.

Ausschalten

Zum Ausschalten des ganzen Systems muss **zuerst der rote Schalter** auf dem Tisch gedrückt werden. Nach einer kurzen Wartezeit kann der **Netzschalter umgelegt**, also der Strom am Tisch

ausgeschaltet, werden. Es empfiehlt sich immer **zuerst** den Pi herunterzufahren und dann den Strom auszuschalten, um Schäden zu vermeiden.

Taster auf dem Tisch

Grüner Schalter: Dieser setzt die Erkennung **zurück** auf Start, d.h. alle LEDs leuchten wieder rot und der Packprozess fängt von vorne an. Dies kann genutzt werden, wenn ein Griff in die Box nicht richtig erkannt wurde.

Roter Schalter: Damit wird der Rechner **runtergefahren**, also die Erkennung ausgeschaltet

Bedeutung LEDs

Für jede Box wird automatisch eine LED bestimmt, die möglichst mittig zur Box liegt. Diese LED kann die Farben rot, gelb und grün annehmen. Links und rechts davon leuchten die LEDs bei entsprechenden Events blau oder sind aus.

Rot:	Aus dieser Box wurde noch **nichts** entnommen
Gelb:	Aus der Box wurde schon etwas entnommen, aber es **fehlen** noch weitere Entnahmen
Grün:	Aus dieser Box ist **alles** entnommen
Blau:	Box ist **leer**
Startsequenz:	Rote LEDs laufen von außen auf die Mitte des LED-Strip und signalisieren, dass der PI erfolgreich gebootet wurde.

Abschlussanimation, alle LEDs blinken: Der Arbeitsvorgang wurde erfolgreich abgeschlossen alle Teile wurden entnommen. Danach kann der Packprozess von vorne begonnen werden. Während der Abschlussanimation ist die Erkennung pausiert.

Anzeige auf dem Monitor

Wie oben gezeigt, ist vor allem für Demo und Testzwecke eine Verwendung eines Monitors nützlich. Außerdem können dadurch die gewählten Einstellungen schnell überprüft und ggf. angepasst werden. Da die Visualisierung aber Rechenleistung in Anspruch nimmt, sollte man entweder das Visualisierungs-Fenster klein ziehen (Kamera liefert eine 1280x720 Auflösung) oder nach ein paar erfolgreichen Durchläufen deaktivieren.

Einstellen von Settings, Anlegen von Aufträgen

Aufträge und Settings werden in JSON Dateien festgehalten. Standardmäßig wird der Auftrag in

`/home/ambos/workspace/Ambos-Pi/data/order/current.json` verwendet.

Allgemeine Settings (v1) sind in `/home/ambos/workspace/Ambos-Pi/data/settings/settings.json`,

Hand-Detektor spezifische Settings (v1) in `/home/ambos/workspace/Ambos-Pi/data/settings/model_settings.json` festgelegt.

Wie ändert man einen **Auftrag** ? Der gestartete Webserver besitzt eine Route zum Updaten der Aufträge (`/api/updateOrder/<uuid>`). Im Body wird der komplette Auftrags-String (JSON als

String) erwartet, welcher dann die Einstellung in `current.json` überschreibt. Es kann natürlich, aber auch manuell die Datei, ohne Webserver, verändert werden. In v2 wird ein Auftrag in der Web/ControlApp angelegt. Infos zu Aufträgen gibt es hier.

Wie ändert man **Settings**? Für die Änderung von Settings ändert man einfach die entsprechenden JSON Attribute in den Dateien ab. Wo welche Settings zu finden sind, zeigt die untenstehende Tabelle. In AmbosV2 liegen alle Settings in einer JSON Datei unter `~/data/v2/settings/settings.json`. Informationen zu den Settings gibt es hier.

Wie kann ich Boxen **einzeichnen**? Über die Route `/api/Image/<int:pid>.jpg` kann das aktuelle Frame der Kamera abgefragt werden. Anschließen können darin Boxen mit einem beliebigen Tool eingezeichnet und die Koordinaten in die Order-Datei geschrieben werden. In v2 wird über MQTT der aktuelle Frame abgefragt.

Wichtig: Aufträge und Settings können zwar während dem Betrieb geändert werden, benötigen aber einen Neustart der Ambos Applikation. Änderungen über den Webserver führen automatisch zu einem Neustart der Applikation, da diese beim Aufrufen der API zunächst gestoppt und danach wieder gestartet wird.

Manueller Neustart

Auf dem Desktop befindet sich ein Skript `Ambos_Restart` welches zunächst den Ambos Prozess und Python (Webserver) stoppt und dann erneut startet. Wer möchte kann zuvor die geöffneten Terminals über *Strg +C* oder mit dem Kreuz oben rechts am Terminal schließen.

Aufträge

Der aktuelle Auftrag ist in der

`/home/ambos/workspace/Ambos-Pi/data/order/current.json`

Datei zu hinterlegen. Diese Datei wird der Ambos Applikation beim Start als Argument übergeben. Es ist also einfacher darin Änderungen vorzunehmen, anstatt eine neue Datei anzulegen und den Programm Parameter abzuändern. Ein Auftrag setzt sich aus der `BoxList` und dem `Order` Objekt zusammen:

Die `BoxList` ist eine Liste aus Box-Objekten, welche die Informationen der eingezeichneten Boxen enthalten:

Box Attribut	Datentyp	Bedeutung
componentID	Integer	Referenz-Id auf die in der Box enthaltene Komponenten (Bauteil). Eine Liste an Komponenten ist im Order-Objekt enthalten. **Wichtig:** die *componentID* bezieht sich immer auf die Position der jeweiligen Komponente in der Komponenten-Liste im Order Objekt.
height	Integer	Höhe der Box
boxID	Integer	ID der Box
width	Integer	Breite der Box
x	Integer	X-Koordinate der linken oberen Ecke der Box
y	Integer	Y-Koordinate der linken oberen Ecke der Box

Das `Order` Objekt enthält neben der Komponenten-Liste noch die String-Felder *orderClient*, *orderIDNumber* und *orderName*, welche zwar eingelesen werden, aber noch nicht wirklich verwendet werden. Ein Komponenten-Objekt besteht aus folgenden Attributen:

Komponenten Attribut	Datentyp	Bedeutung
CompGrp	String	Komponenten Gruppe
name	String	Höhe der Box
quantity	Integer	Welche Anzahl der jeweiligen Komponente für den aktuellen Auftrag benötigt wird

Auf dem Image ist bereits eine Beispiel Konfiguration der *current.json* Datei enthalten.

Settings

Wie oben erwähnt, gibt es allgemeine und Modell-spezifische Settings. Settings für V1 liegen unter

`/home/ambos/workspace/Ambos-Pi/data/settings/*`

und können dort direkt mithilfe eines Text-Editors (z.B. Pluma) geändert werden. Dafür sind eine Maus und Tastatur erforderlich.

Wichtig: Die JSON Struktur muss erhalten bleiben und darf keinen Fehler aufweisen (z.B. fehlende Kommas).

Wichtig: Für V2 sind beide Settings-Dateien zusammengelegt. Diese enthalten aber vorwiegend Default-Werte, welche mit den Settings, empfangen über MQTT, überschrieben werden.

Die einzelnen Einstellungen und ihre Bedeutung sind in folgender Tabelle aufgeführt. Die Beispielwerte sind gleichzeitig als empfohlene Startwerte zu verstehen. (Es sind auch direkt zusätzliche Paramater für AmbosV2 hinterlegt)

Parameter	Funktion	Datei (V1)	V1	V2
red_offset, green_offset, blue_offset **Integer** **z.B. 0, 2, 1**	Sollte der LED-Streifen nicht die gewünschten Farben anzeigen, liegt das daran, dass es RGB und BGR LED-Streifen gibt. Mit diesen Parametern kann dies angepasst werden. Es werden für alle drei Parameter die Werte 0, 1 und 2 erwartet, wobei jeder exakt einmal für eine korrekte Funktionsweise vergeben sein muss.	settings	✓	✓
threshold_change **Integer- Liste** **z.B. [10]**	Schwellwert für die Erkennung von Änderungen in einer Box. Die Erkennung wird bei kleineren Werten sensitiver. Sollte es zu vielen fehlerhaften Erkennungen kommen, kann dieser Parameter langsam erhöht werden, bis dies nicht mehr auftritt. Liste sollte entweder pro Box oder nur einen Wert, stellvertretend für alle Boxen, enthalten.	settings	✓	✓
threshold_empty_edge **Float-Liste** **z.B. [0.3]**	Schwellwert für die Erkennung von leeren Boxen mithilfe Kantenerkennung. Ist als prozentualer Wert zu verstehen und legt fest, wieviel Prozent der Pixel zu Kanten gehören dürfen, sodass eine Box noch als leer erkannt wird. Je kleiner, desto weniger Kanten dürfen in einer leeren Box zu sehen sein, d.h. desto später wird die Box als leer erkannt. Liste sollte entweder pro Box oder nur einen Wert, stellvertretend für alle Boxen, enthalten.	settings	✓	✓
threshold_empty_color **Float-Liste** **z.B. [0.2]**	Schwellwert für die Erkennung von leeren Boxen mithilfe Farbsättigung. Ist als prozentualer Wert zu verstehen und legt fest, wieviel Prozent der Pixel dieselbe Farbe enthalten müssen, sodass eine leere Box erkannt wird. Je	settings	✓	✓

	größer, desto mehr freie Fläche muss zusehen sein, d.h. desto später wird die Box als leer erkannt. Liste sollte entweder pro Box oder nur einen Wert, stellvertretend für alle Boxen, enthalten.			
num_led, offset_ledstart, offset_ledend **Integer** **z.B. 24, 85, 1195s**	Die Anpassung der LED-Streifen auf die Boxen erfolgt mit diesen Parametern. Num_led gibt die Anzahl der LEDs des LED-Streifens an. Die Offsets beschreiben das linke und rechte Ende des LED-Streifens im Bild, gemessen an Pixelpositionen. D.h. wenn *offset_ledstart* den Wert 50 hat, wird davon ausgegangen, dass der LED-Streifen nach 50 Pixeln im Bild beginnt. Das Bild ist insgesamt 1280 Pixel breit. Ebenso müssen perspektivische Verzerrungen durch Höhenunterschiede zwischen LED und Boxen beachtet werden.	settings	✓	✓
reset_pin, shutdown_pin **Integer** **z.B. 6, 5**	Die gewählten Pins können bei Bedarf vertauscht werden oder auf einen nicht belegten Pin gesetzt werden, um sie zu deaktivieren. Bei Deaktivierung des Shutdown_Pin unbedingt über einen angeschlossenen Monitor das System herunterfahren, andernfalls kann es zu einem beschädigten System kommen.	settings	✓	✓
enable_visualization **Bool** **z.B. true**	Die Visualisierung kann hier optional abgeschaltet werden, was zu einer schnelleren Erkennung führt. Zu Test- und Demozwecken ist diese allerdings hilfreich.	settings		
enable_hand_detection **Bool** **z.B. true**	Die neu eingebaute ML-Handerkennung kann über diesem Parameter ausgeschalten werden, um zum ursprünglichen Verfahren zurückzukehren (nicht empfohlen)	settings	✓	✓
rotate_image **Bool** **z.B. false**	Sollten die Boxen zwischen App und Pi gedreht erscheinen, kann dies hier geändert werden. *Es wird nur eine Drehung um 90° im Uhrzeigersinn berücksichtigt.*	settings	✓	✓
debug_text **Bool** **z.B. true**	Es werden weitere Ausgaben am Konsolenfenster angezeigt zur Fehlerfindung und Veranschaulichung	settings	✓	✓
led_file **String** **z.B. „blink.led“**	Die Abschlussanimation nach Vervollständigung einer Packung kann mit diesem Parameter angepasst werden. Im Ordner */home/workspace/Ambos-Pi/data/led* finden sich noch weitere Beispiele. Auch eigene können nach demselben Muster erstellt werden. Es werden immer absolute Pfade in Anführungszeichen erwartet.	setting	✓	✓
empty_detection_via_color **Bool-Liste** **z.B. [false]**	Entscheidet über die Methode, wie leere Boxen erkannt werden Je nach Bauteil, bietet sich die Sättigungs- (*true*) oder Kantenerkennung (*false*) an. Liste sollte entweder pro Box oder nur einen Wert, stellvertretend für alle Boxen, enthalten.	settings	✓	✓
ambos_version **Integer** **z.B. 1**	Entweder 1 oder 2. In 1 übernimmt die Ambos Applikation selbst die Prozessteuerung. In v2 erhält sie lediglich Anweisungen über MQTT	settings	✓	✓
pause_detection_threshold **Integer** **z.B. 3**	Sobald eine Kontur im Bild erkannt wird, schaltet sich die Handerkennung (falls aktiviert) ein. Über diesen Wert wird festgelegt, nach wie vielen Frames mit Händen, die Handerkennung wieder ausgeschalten werden kann. Dies spart Ressourcen und ist effizient. Der Zähler wird auf 0 gesetzt, sobald keine Kontur mehr erkannt wird und erneut inkrementiert, sobald wieder Hände erkannt werden	settings	✓	✓
min_number_detections **Integer** **z.B. 1**	Sobald eine Kontur im Bild erkannt wird, schaltet sich die Handerkennung (falls aktiviert) ein. Ein Zähler wird inkrementiert, sobald eine Hand erkannt wird. Nachdem keine Kontur mehr erkannt wird, bestimmt dieser Wert, in wie vielen Frames eine Hand erkannt werden musste,	settings	✓	✓

	sodass ein Griff überhaupt stattgefunden haben kann. Bei sehr schnellen Griffen sollte die Zahl lieber auf 1 gesetzt werden.			
sounds_on **Bool** **z.B. true**	Aktiviert/Deaktiviert Töne bei Events	settings	✓	✓
model_classes_file **String**	Klassendatei für den Yolo Handdetektor (sollte nicht geändert werden)	model_settings	✓	✓
model_input_size **Integer** **z.B. 128**	Eingabegröße der Bilder für den Handdetektor. Muss zwingend teilbar durch 32 sein! Je kleiner desto schneller die Verarbeitung aber je ungenauer die Erkennung. Sollten schnelle Griffe nicht erkannt werden, kann diese z.B: auf 96 reduziert werden	model_settings	✓	✓
model_cfg **String**	Yolo Config Datei Pfad	model_settings	✓	✓
model_weights **String**	Yolo Gewichte Datei Pfad	model_settings	✓	✓
model_conf_thresh **Float** **z.B. 0.4**	Schwellwert für die Confidence des Modells. Je größer, desto sicherer muss sich das Modell sein. Bei falschen Erkennungen, kann probiert werden, diesen Wert langsam zu erhöhen. (Hängt aber davon ab, mit welcher Genauigkeit mögliche nicht-Hand Objekte erkannt werden)	model_settings	✓	✓
model_nms_thresh **FLoat** **z.B. 0.1**	Schwellwert für die Non-max surpression um doppelte Erkennungen zu löschen. Falls eine Hand doppelt erkannt wird, sollte dieser Wert erhöht werden. Aber prinzipiell spielt es keine Rolle, ob eine oder mehrere Hände erkannt werden	model_settings	✓	✓
grayscale **Bool** **z.B. false**	Soll das Kamerabild als Graustufenbild auf das Modell gegeben werden? Macht nur Sinn, wenn das entsprechende Modell damit trainiert wurde	model_settings	✓	✓
padding **Integer** **z.B. 20**	Fügt um die Boxen jeweils *padding* Pixel in jede Richtung hinzu. Dieses vergrößerte Bild wird dann auf das Modell gegeben. Vor allem dann wichtig, wenn nur Fingerspitzen in die Box hineinragen.	model_settings	✓	✓
padding_front **Integer** **z.B: 0**	Ist die Kamera gegenüber dem Arbeiter platziert, so vergrößert *padding_*front zusätzlich das Eingabebild auf das Modell in Richtung der Boxöffnung. Hilfreich wenn es für sehr kleine Teile ausreicht, nicht weit in die Box greifen zu müssen.	model_settings	✓	✓
workspace_id **Integer** **z.B. 0**	Id des Arbeitsplatzes. Damit versucht sich der Prozess bei der *ControlApp* zu registrieren/einzuloggen. MQTT Nachrichten werden nur verarbeitet, wenn diese Id enthalten ist	-	×	✓
workspace_name **String** **z.B. „Arbeitsplatz 1“**	Name des Arbeitsplatzes. Wird seitens der *ControlApp* verwendet um die Arbeitsplatz-Json anzulegen.	-	×	✓
mqtt_id **String** **z.B. „PI_Client“**	MQTT Client Name	-	×	✓
mqtt_sendTopic **String** **z.B. „SENSOR_MSG“**	Topic für gesendete Nachrichten zur *ControlApp* (nicht ändern!)	-	×	✓
mqtt_receiveTopic **String** **z.B. „CONTROL_MSG“**	Topic für empfangen Nachrichten von *ControlApp* (nicht ändern!)	-	×	✓
mqtt_port **Integer** **z.B. 1883**	MQTT Broker Port	-	×	✓
mqtt_host **String** **z.B. „localhost“**	Broker Host für Verbindung	-	×	✓
mqtt_keepalive **Integer**	KeepAlive Sendezeit	-	×	✓

z.B. 60				

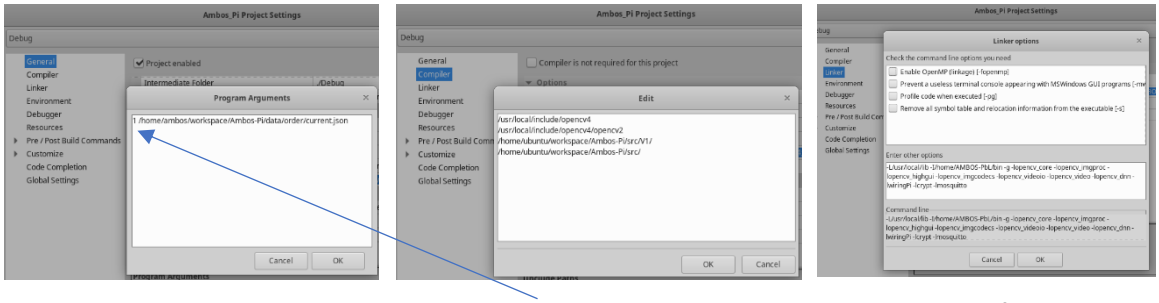
Für die Weiterentwicklung

Folgende Informationen sollen dein Einstieg in die Weiterentwicklung erleichtern. Der Code ist bereits dafür ausgelegt, zwischen AmbosV1 und AmbosV2 zu wechseln. Lassen Sie sich also nicht davon verwirren, wenn bspw. MQTT Code auftaucht. Zuerst ein paar generelle Sachen:

Bearbeitung und Start über Codelite

Auf dem Image befindet sich der Editor Codlite für das Programmieren in C++. Die Workspace-Datei, welche mit dem Editor geöffnet werden sollte, ist `/home/workspace/Ambos-Pi/Ambos_Pi.workspace` (sollte beim Starten von Codelite bereits offen sein).

Im Editor sind bereits alle notwendigen **Include-Paths**, **Linker Options** und **Program Arguments** gesetzt, sodass Ambos direkt aus Codelite über `Build>Build Project` gebaut und `Build>Run` gestartet werden kann. Für Ergänzung gelangt man über ein Rechtsklick auf das Projekt (Ambos_Pi) in die Settings:



Wichtig: Das erste Programm Argument ist die **Ambos Version**. Setzt man dieses auf 1, so wird zusätzlich der Pfad zur **Order**-Datei als Argument erwartet. Bei AmbosV2 gibt es keine Order Datei, da der Auftrag über die ControlApp angelegt und gesteuert wird.

Codelite ist so konfiguriert, dass es kompilierte Dateien in einem Debug-Ordner ablegt. Für das einfache Entwickeln bietet es sich also an, die Applikation nicht über den Webserver zu starten, sondern einfach direkt in Codelite. Natürlich kann aber auch einfach mit einem Texteditor programmiert und manuell kompiliert werden.

Automatischer Start nach dem Boot über den Webserver

Nach dem Boot des Pis wird folgender Befehl (über das Skript `StartWebserver`) automatisch ausgeführt:

```
python 3 /home/ambos/workspace/Ambos-Pi/bin/server/webserver.py
```

D.h. die Ambos Applikation wird bei automatischer Ausführung immer über einen kleinen Webserver gestartet. Bevor der Webserver Ambos startet, liest er aus `/home/ambos/workspace/Ambos-Pi/data/settings/settings.json` die gesetzte Ambos Version. Die Version kann aber auch direkt, ohne das Einlesen einer Datei, gesetzt werden. Der Pfad zur Order-Datei ist ebenfalls in der Python-Webserver Datei definiert - damit sind die notwendigen Programm-Arguments festgelegt. Dann wird Ambos über eine bestimmte kompilierte Datei gestartet (bei v2 ohne Order-Datei):

```
{Home_DIR}/bin/AMBOS_Pbl {VERSION} {OrderDatei}
```

Damit als Änderungen über Codelite auch die automatisch gestartete Version betreffen, muss der Code manuell zu `AMBOS_Pbl` kompiliert werden. Dafür führt man folgendes Skript aus `/home/ambos/workspace/compile`, welches den Kompilier-Befehl enthält:

```
g++ -std=c++11 src/*.cpp  -I/usr/local/include/opencv4/opencv2 -
I/usr/local/include/opencv4 -L/usr/local/lib -I/home/AMBOS-PbL/bin -
g -lopencv_core -lopencv_imgproc -lopencv_highgui -lopencv_imgcodecs
-lopencv_videoio -lopencv_video -lopencv_dnn -lwiringPi -lcrypt -
lmosquitto  -o "bin/AMBOS_PbL"
```

Wichtig: Der Webserver startet momentan auf 0.0.0.0 und Port 5000 und startet direkt den MQTT Broker, falls Version 2 aktiviert ist. In Version 2 wird (vermutlich) die *ControlApp* und *WebApp* diesen Webserver ersetzten

Sounds

In `/home/ambos/workspace/Ambos-Pi/data/sounds` befinden sich 4 Dateien:

- start.wav => ertönt beim Starten
- grip.wav => ertönt nach erkannter Entnahme
- empty.wav => ertönt nach Erkennung einer leeren Box (einmal pro Wechsel)
- full.wav => ertönt nach Erkennung einer vollen Box (einmal pro Wechsel)

Für Änderungen platziert man einfach die gewünschte WAV Datei in den Ordner und benennt sie entsprechend ihrer Verwendung um. Die Namen sollten gleich gewählt werden, da der Pfad im Code fest hinterlegt ist.

Aktivierung von Tönen über MQTT für V2 ist noch nicht umgesetzt (TODO).

Grundsätzlicher Ablauf

Aus jedem Frame werden zunächst die einzelnen Box Bereiche ausgeschnitten und als Graustufen Bild den einzelnen Boxobjekten zugeordnet (hier als *Hintergrundbild* bezeichnet). Sequenziell werden für jeden Boxausschnitt dann folgende Schritt vollzogen:

- Zunächst wird der **statische** Hintergrund in den Bildern entfernt. Danach werden **Konturen** detektiert. Bewegt sich etwas im Boxbereich, z.B. eine Hand, dann werden Konturen erkannt.
- Wird eine Kontur erkannt, so schaltet sich die **Handerkennung** ein und erhält den aktuellen **Boxausschnitt** (in Farbe oder Graustufen), zuzüglich einem **Padding** und skaliert auf die gesetzte **Dimension** als Eingabe. Sobald eine Hand erkannt wurde, wird ein **Zähler** inkrementiert. Dieser wird wieder auf **0** gesetzt, sobald **keine Kontur** mehr erkannt wird
- Ist nun **keine** Bewegung mehr im Bild, wird **keine Kontur** mehr erkannt. Bevor der oben genannte Zähler auf 0 gesetzt wird, wird geschaut, ob und in wie vielen Frames eine Hand erkannt wurde. Ist die Anzahl größer als ein Schwellwert, dann (und nur dann! (zumindest nur wenn die Handerkennung an ist)), wird das aktuelle Box-Bild mit dem Bild vor der letzten Konturerkennung verglichen**. Wichtig:** Das *Hintergrundbild* der einzelnen Boxen wird also nur geupdatet, wenn **keine bewegte Kontur** im Bild ist! Während Bewegungen im Bild wird das *Hintergrundbild* nicht durch das aktuelle ersetzt! Ein Griff wird dann erkannt, wenn der Unterschied zwischen dem aktuellen Bild und dem alten *Hintergrundbild* der Boxen ausreichend groß ist.

- Wird ein Griff erkannt, so wird basierend auf den aktuellen Box-Bildern versucht den Füllstand zu erkennen (entweder mittels Kantenerkennung oder Sättigungs-Histogramm => leere Boxen weisen weniger Kanten und einheitliche Farbbereich auf)

Ob erkannte Griffe mit dem aktuellen Auftrag vereinbar sind, wird entweder direkt von Ambos (v1) oder von der *ControlApp* (v2) entschiedenen. In beiden Fällen werden als Konsequenz entsprechende LEDs gesetzt.

Startpunkt im Code

Wie schon erwähnt, ist der Code darauf ausgelegt, für beide Versionen zu funktionieren. Deshalb mag er an mancher Stelle etwas unnötiger kompliziert aussehen. Dies beruht aber darauf, dass eine Umstellung auf v2 möglichst einfach sein soll.

Der Einstiegspunkt befindet sich in `AMBOS_Pbl_Main.cpp`. Hier sind die Pfade zu den Settings-Dateien hinterlegt. Abhängig von der Version werden die entsprechenden Settings eingelesen und die zentralen Objekte damit initialisiert. Diese sind:

Für die Prozess Steuerung

Version1: Die Prozesslogik sitzt in der `Order.cpp`. Hier wird die Verarbeitung der Frames veranlasst, die LEDs gesetzt und auf Taster-Events reagiert. Die **Main-Loop** des Programms sitzt hier! Während des Betriebs werden gewisse Flags in den Boxobjekten gesetzt (z.B. wenn alle notwendigen Komponenten aus einer Box entnommen worden sind). Diese Flags werden dann genutzt, um die LEDs entsprechend farblich zu aktivieren.

Version2: Da Version2 unabhängig von Order-Objekten läuft und nur erkannte Griffe / leere Boxen meldet, liegt hier die **Main-loop** in der Klasse `AmbosV2.cpp`. Hier wird die Verarbeitung der Frames veranlasst, auf Taster-Events reagiert und MQTT-Nachrichten (v.a. Verarbeitung von empfangenen Settings und LED-Anweisungen) abgearbeitet.

Framehandler

Der `FrameHandler` ist das zentrale Objekt der Bildverarbeitung. Hier werden die Kamerabilder ausgelesen und auf die Boxbereiche zugeschnitten. Die im vorherigen Abschnitt beschriebenen Verarbeitungsschritte werden in den Funktionen `checkBoxContourAndHand`, `checkBoxChange` und `checkBoxEmptiness` durchgeführt. Zusätzlich wird dem Framehandler entweder ein `Order` Objekt (v1) oder ein `MQTT` Objekt (v2) übergeben. Mit ersterem können bei erkannten Griffen die Anzahl der entnommenen Teile erhöht und automatisch fertig Aufträge erkannt werden. Letzteres wird verwendet um bei Events (leere/volle Box, Griff) eine MQTT-Nachricht zu publishen.

Für das Einlesen der Json-Dateien wird `rapidjson` verwendet. Mit diesen Informationen und den Code Kommentaren sollte, beginnend bei `AMBOS_Pbl_Main.cpp`, die Einarbeitung in den Code leichter fallen.

DrawBoxes

Unter `/home/ambos/workspace/scripts` liegt ein rudimentäres Skript zum Einzeichnen von Boxen. Das Programm erwartet als Argument einen Pfad zu einer existierenden Datei und legt dort die eingezeichneten Boxen im JSON Format ab. Also Key wird `BoxList` verwendet. Es ist möglich

direkt den Pfad der Order-Datei aus Ambos als Argument zu übergeben. Die darin enthaltende `BoxList` wird dann abgeändert bzw. neue Boxen werden hinzugefügt

Wichtig: Das Skript setzt per Default für alle Boxen, egal ob schon vorhanden oder neu angelegt, eine `componentID` von 0. D.h. wird direkt die Order-Datei angegeben, müssen die `componentID` s nachträglich eingetragen werden.

Befehl zum Kompilieren
```
./drawBoxesCompile
```

Befehl zum Ausführen
```
./drawBoxes ../Ambos-Pi/data/order/current.json
```

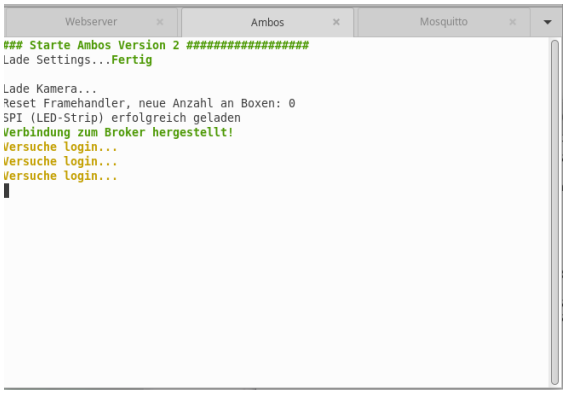
Verwendung
Nach dem Start sollte das aktuelle Kamerabild auftauchen. Die Boxen können immer noch justiert werden. Im Folgenden werden die Schritte erklärt um 2 Boxen einzuzeichnen:

- Taste `f` drücken => Kamerabild freezed
- `0` drücken => BoxId wird auf 0 gesetzt
- Einzeichnen der Box mit der Maus (mit `d` löscht man die aktuelle Zeichnung)
- `s` drücken => Box wird mit den Koordinaten abgespeichert
- `1` drücken => BoxId wird auf 1 gesetzt und vorherige Zeichnung wird entfernt
- Einzeichnen der Box mit der Maus (mit `d` löscht man die aktuelle Zeichnung)
- `s` drücken => Box wird mit den Koordinaten abgespeichert
- 2x`ESC` zum Beenden und speichern.

Für AmbosV2

Folgende Informationen sind für zukünftige Arbeiten an der zweiten Version von Ambos gedacht.

- Startet man Ambos in Version 2 über den bisher verwendeten Webserver, dann öffnen sich drei Terminals: Eins für den Webserver, eins für den MQTT-Broker und eins für die Applikation selbst



- Beim Start der Applikation wird versucht sich mit dem MQTT Broker zu verbinden. Gelingt dies nicht, befindet sich der Prozess solange in einer **blockierenden** Schleife, bis die Verbindung aufgebaut werden konnte. Gleiches gilt für einen Verbindungsabbruch während dem Betrieb

- Steht die Verbindung zum Broker, läuft die **Loop** für die Erkennung – da aber ohne eine MQTT Settings - Nachricht keine Boxen hinterlegt sind, passiert erstmal nichts
- Der Prozess versucht sich dann mithilfe einer **Login**-Nachricht bei der *ControlApp* zu melden. Empfängt er eine an ihn gerichtete MQTT Nachricht (über die Workspace ID), dann gilt der Prozess als **eingeloggt** und ist damit auch bei der *ControlApp* hinterlegt.

- Mithilfe einer **Settings-Nachricht** werden alle relevanten Einstellungen und die Boxen an die SensorApp (hier Ambos) übermittelt. Wichtig ist hierbei das Feld `resetRequired`**:** Basierend auf den Settings werden die initial angelegten Objekte (FrameHandler, Boxen, Hand-Detektor…) geupdatet. Es gibt bestimmte Attribute, die für jede Box einen Wert speichern. Diese werden bei einem Reset zurückgesetzt und auf die aktuelle Anzahl an Boxen angepasst. Enthält nun eine Settings-Nachricht mehr Boxen als zuvor oder Boxen mit anderen IDs und `resetRequired` ist *false*, dann würde es zu Index Fehlern kommen (wird aber abgefangen!). Ein *Reset* ist nicht notwendig, wenn man die gleichen Boxen nur verschiebt oder andere Parameter ändert.

- Zwar könnte der Pi auch MQTT **Threading**, dies wird aber nicht benötigt, da während der Verarbeitung von Nachrichten die Erkennung nicht aktiv sein muss, da entweder Settings geändert werden (hier macht es erst Sinn danach weiterzuarbeiten) oder das aktuelle Bild abgefragt wird (auch hier ist die Annahme, dass dies nur geschieht wenn man etwas grundlegendes am Prozess ändern möchte)
- Die Ambos Applikation enthält über MQTT die Anweisung zum **Setzen** der LEDs. Da dies aber auch nur einzelne Boxen betreffen kann und nicht den Status aller LEDs übermittelt, werden **Sets** genutzt, um den Zustand aller LEDs festzuhalten. Außerdem wird bis zum endgültigen Schreiben des LED-Strings nur die `BoxIds` und nicht direkt die LED_POSITION verwendet, da diese sich ggfs. durch eine Settings-Nachricht verändern kann (wenn z.B. `led_start` geändert wird)

Beispiele für eine Settings-Nachricht und Nachrichten zum Setzen der LEDs sind in `/home/ambos/workspace/Ambos-Pi/examplesMqtt.txt` hinterlegt