
# Titel der Arbeit

Diplomarbeit

zur Erlangung des akademischen Grades
Diplominformatiker

**HUMBOLDT-UNIVERSITÄT ZU BERLIN**
**MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT II**
**INSTITUT FÜR INFORMATIK**

eingereicht von: Ronald Klaus
geboren am: 5. August 1980
in: Hennigsdorf

Gutachter(innen): ...
...

eingereicht am: ..... verteidigt am: .....

**Abstract**

In dieser Arbeit wird ein Tracking-Detection-Algorithmus hinsichtlich seiner Eignung für das Finden und Verfolgen von Flugroboter, wie Quadro- und Hexakopter, untersucht. Im Detail besteht die Aufgabe, dass ein solcher Roboter ausgestattet mit einer Kamera, visuellen Informationen für das Finden eines anderen Roboters verarbeitet.

Der untersuchte Ansatz TLD - Tracking Learning Detection [TLD] ist ein semi-automated single-target tracking-Algorithmus. Das single-target tracking schätzt in einer Sequenz von Bildern $I_0, \ldots, I_m$ den Status $x_k$ eine Objekts im Bild $I_k$ [VID]. Der semi-automated-Ansatz bezieht sich auf die Initialisierung des Trackens. Im Gegensatz zum automated tracking, in dem der Tracker mittels bereits bestehende Informationen initialisiert wird, und dem manual tracking, bei dem eine Nutzerinteraktion in jedem Bild nötig ist, wird im semi-automated-Tracking ein eine Nutzereingabe für die erste Objektdefinition benötigt.

Für eine erfolgreiche Abreitsweise jedes Trackers, egal welchen Ansatz im zugrunde liegt, sind zwei Anforderungen essentiell: Genauigkeit und Robustheit [PYR]. Beide sind jedoch, bezogen auf die Möglichkeiten zur Optimierung, gegensätzlich. Die Genauigkeit wird vor allem dadurch erreicht wird, dass der Suchradius des Objekts im Folgebild möglichst gering ist, um eine Fehlerfunktion zu minimieren. Zur Optimierung der Robustheit und damit die Reaktion auf eventuelle Änderungen der Lichtverhältnisse oder Bewegungen des Objekts, sollte gerade ein großer Radius zur Suche genutzt werden, damit das Objekt beispielsweise durch eine schnelle Bewegung nicht aus dem Suchfeld verschwindet. Damit muss ein Mittelweg gefunden werden, da eine gleichzeitige Optimierung beider Anforderungen nicht möglich ist.

Typischerweise ist das Finden und Verfolgen von Objekten mit einer nichtstatitischen Kamera in der "realen" Welt eines der herausfordernsten Problemstellungen der Computer Vision, da die Umwelt einen großen Einfluss auf die Qualität der visuellen Daten hat. Lichtverhältnisse variieren, eine Fülle anderer Objekte hat eventuell große Ähnlichkeit zum gesuchten Objekt haben, komplexe und nichtstatische Strukturen wie Bäume erschweren das Separieren. Zusätzlich ist definiert, dass das Objekt, in diesem Fall der Kopter, keinerlei Markierungen in Form von farbigen Markern oder speziellen, eindeutigen Mustern erhält.

# Contents

# 1 Einleitung

**Computer Vision**, das machinelle Sehen, ist eine große Herausforderung in der Bildverarbeitung und Feld intensiver Forschung. Die zunehmende Leistungsfähigkeit der Rechner und verbesserte Qualität auch preiswerter Kameratechnik in den letzten Jahren intensivieren die Bestrebungen auf diesem Gebiet zusätzlich. Die dadurch entstandenen Ansätze, Modelle und Methoden haben längst in vielen Bereichen des Alltags Einzug gehalten. Kaum ein Auto wird ohne "intelligente" Sensorik ausgestattet und unterstützt den Fahrer beispielsweise durch Lesen von Straßenschildern, Erkennen von Fahrbahnmarkierungen sowie Fußgängern oder andere Verkehrsteilnehmern [PED]. Die Industrie nutzt computergesteuerte Auswertungen von Kamerabildern zur Qualitätssicherung von Bauteilen wid zum Beispiel Displays [LCD]. In der Medizin helfen computergestützte visuelle Verfahren bei der Analyse von Bilddaten und unterstützen so die Diagnostik [MIP]. Und nicht zuletzt bildet die Auswertung von Kameradaten auch im Forschungsbereich der Robotik oft die Grundlage für viele Unterschiedliche Anwendungen. Die Aufgaben, die an den Bereich Computer Vision gestellt werden, sind somit sehr vielfältig.

Diese Arbeit behandelt einen speziellen Teilbereich des machinellen Sehens: das Finden und Verfolgen von Objekten. Ein definiertes Objekt, das in einer bestimmten Art und Weise repräsentiert ist, soll somit in einem Bild gefunden (Detection) und verfolgt (Tracking) werden. Da noch nicht verstanden ist, wie z.B. das menschliche Gehirn visuelle Daten zur Erkennung von "gelernten" Objekten verarbeitet, ist eine Adaption auf Computerebene nicht möglich. Die Natur kann als nicht vollkommen als Vorbild dienen. Aus diesem Grund wurden im Laufe der letzten Jahrzehnte unterschiedliche Ansätze entwickelt, die mittlerweile sehr gute Ergebnisse liefern, sich allerdings oft auf einen Teilbereich oder ein spezielles Szenario beschränken.

Neben der algorithmischen Lösung der Objekterkennung treten zudem eine Reihe technischer Herausforderungen auf, die beinahe jeder Tracking-Detection-Ansatz zumindest teilweise bewältigen muss:

- Informationsverlust durch Projektion der 3D-Welt in 2D-Bilder
- Bildrauschen
- komplexe Bewegungen des Objekts
- teilweises oder komplettes Verdecken des Objekts

- Änderungen in der Belichtung der Szene
- bewegungen der Kamera
- etc.

Umwelteinflüsse und technische Einschränkungen spielen somit eine große Rolle beim Erfolg des Trackens[OTS].

Die Wahl der richtigen beziehungsweise geeignetsten Bildverarbeitungsmethoden richtet sich nicht zuletzt auch nach dem Einsatzgebiet der Implementation. Zusätzlich hat es auch Einfluss auf die Komplexität der Lösung. Ein statische Überwachungskamera, die zum Beispiel Bewegungen registrieren soll, bietet beispielsweise ein softwaretechnisch relativ simple Lösung. Da sie permanent den gleichen Hintergrund aufnimmt, kann durch einfache Subtraktion zweier Folgebilder eine robuste Objekterkennung erfolgen. Das Erkennen von bestimmten Objekten wie Gesichtern, Straßenschildern oder Produktionsfehlern benötigt wiederum andere, komplexere Methoden, da die Spezifität des Objekts eine Rolle spielt. Manche Aufgabenstellungen sind mittlerweile gut und zufriedenstellend zu lösen, anderen hingegen stellen weiterhin eine große Herausforderung dar. Die im Anschluss kurz vorgestellten Paradigmen begegnen den Probleme auf unterschiedliche Arten und Weisen.

## 1.1 Zielsetzung dieser Arbeit

Was soll gemacht werden? Flugroboter wie Quadro- oder Hexakopter sollen sich gegenseitig mittels eines Kamerabild erkennen können, um entsprechende autonome Handlungen auszuführen. Dabei gibt es eine Reihe von Vorgaben, die erfüllt werden müssen.

Die Kopter erhalten keine zusätlichen Komponenten wie Leuchtdioden oder spezielle Farb- oder Musterapplikationen zur Identifizierung erhalten. Jeder Kopter soll seine Individualität durch sich selbst ausdrücken und so zu identifizieren sein.

Das Detektieren der Kopter soll unter realen Bedingungen funktionieren. Umwelteinflüsse, Veränderungen der Lichtverhältnisse etc. müssen dementsprechend behandelt und gefiltert werden. Außerdem soll die Verarbeitung in Echtzeit erfolgen, weil die Ergebnisse direkt Einfluss auf die Navigation der Flugroboter haben.

Diese Vorgaben schränken die Wahl der Methoden stark ein, da Algorithmen, die zum Beispiel auf die Verarbeitung von Farbwerten basieren, nicht in Betra-

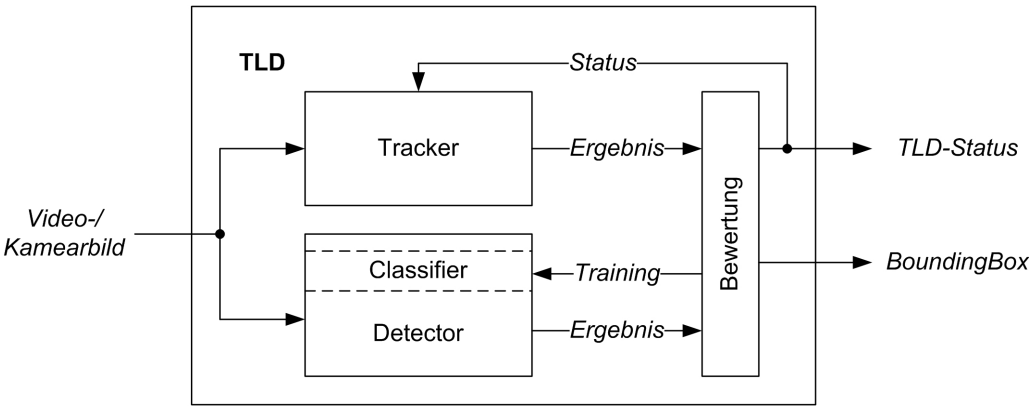


Figure 1: TLD-Schema

cht kommen, oder andere Verfahren zu rechenintensiv für eine Echtzeitberechnung sind.

Deshalb wurde zur Umsetzung der online-tracking-detection-Algorithmus *Tracking-Learning-Detection* (TLD) gewählt. TLD bietet sich als Ansatz an, weil er beliebige Objekte in einem Bild finden kann, keine speziellen Initialisierungsmechanismen oder Lernphasen benötigt, und relativ gut zu verstehen und zu implementieren ist. Das allgemeine Verarbeitungsschema ist in Abbildung 1 dargestellt.

Als Sprache für die Umsetung wurde C++ gewählt, weil sie aufgrund der Möglichkeit relativ hardwarenah zu programmieren, schnelle Programme ermöglicht und durch OpenCV eine gut dokumentierte und umfangreiche Sammlung von Algorithmen speziell für den Bereich der Computer Vision liefert.

Im Anschluss zur Implementierung werden eine Reihe von Tests durchgeführt, die zu Beurteilung der Eignung von TLD für den speziellen Fall des Koptertrackings dienen. Es wird analysiert, welche Bedingungen zwingend gelten müssen, damit dieser Ansatz erfolgreich genutzt werden kann, und unter welchen Umständen sich TLD nicht eignet.

## 1.2 Aufbau der Arbeit

Die Arbeit ist wie folgt gegliedert.

**Kapitel 1** Das erste Kapitel stellt eine kurze Einführung in den Bereich der Computer Vision dar und leitet kurz den speziellen Bereich der Objekterkennung ein.

Weiterhin wird die Zielsetzung der Arbeit definiert und der genutzte Algorithmus in Grundzügen vorgestellt.

**Kapitel 2** Dieses Kapitel gibt einen kurzen Überblick zu verwandten Arbeiten und verschiedenen Ansätzen zur Lösung des Problems des machinellen Sehens. Unterschiedliche Paradigmen und deren Einsatzgebiete werden kurz vergleichend vorgestellt.

**Kapitel 3** Hier erfolgt die genaue Beschreibung von TLD. Es werden ausführlich die theoretischen und mathematischen Hintergründe vorgestellt und der Algorithmus im Detail vorgestellt.

**Kapitel 4** Im vierten Kapitel wir kurz die C++-Implementierung beschrieben. Zusätzlich werden die verwendeten Testszenarien vorgestellt und die durchgeführten Tests beschrieben.

**Kapitel 5** Den Abschluss dieser Arbeit bildet dieses Kapitel der Auswertung und Analyse er Testergebnisse sowie eine Reihe von Vorschlägen für etwaige Verbesserungen.

# 2 Verwandte Themen (Verwandte Arbeiten?)

Arbeiten, die sich speziell mit dem Tracken von Quadro- oder Hexakopter beschäftigen, gibt es derzeit kaum. Im folgenden Abschnitt werden deshalb prinzipielle Tracking und Detection-Ansätze sowie verschiedene Machine Learning-Methoden vorgestellt. Ein strikte Trennung von Tracking und Detection ist im Übrigen kaum möglich, das aktuelle Algorithmen meist eine Kombination darstellen. Prinzipiell ist in jedem Ansatz auch eine Objektrepräsentation vorhanden, die mittels maschinellem Lernen online beziehnungsweise offline vorbereitet und gelernt werden muss.

## 2.1 Object Tracking

Object-Tracking ist, allgemein gesprochen, das Verfolgen von bewegten Objekten in einer Bildsequenz über die Zeit mittels Bewegungsschätzung. Es gibt verschiene Arten der Objektrepräsentation und Trackingmethoden, die in den letzten Jahren entwickelt wurden. Sie unterscheiden sich zum einen in der Repräsentaiton des Objekts, mittels Konturen, Punkte, Modelle oder optischem Fluss, als auch in der Schätzmethode. Ein verbreiteter und auch in TLD genutzter Ansatz ist das Frame-to-Frame Tracking, bei dem die Position des Objekts im Bild aufgrund der Daten des Vorgängerbildes geschätzt wird. Die genutzten Verfahren sind vielfältig und unterscheiden sich hinsichtlich Objektrepräsentaiton und Methodik.

Beim Template-Tracking [OPT][GFT][KBT] wird ein Objekt als Template (Histogramm oder Patch) erfasst und die Bewegung (Motion) ist als Änderung mit dem kleinesten Missmatch zum Kandidaten für das gesuchte Template definiert. Je nach Realisierung ist das Template statisch[KBT] oder flexibel [OPT][GFT], es kann sich also während des Trackens ändern. Es gibt auch Ansätze, die eine Mischung aus beidem implementieren [TUP][SMAT][RDT] oder auch nur Teile des Templates nutzen [ROAM][RFT].

Um dem Nachteil der eingeschränkten Modellierung durch Templates zu begegnen, wurden weitere Repräsentationen für die Objekte entwickelt, beispielsweise die generative Modellierung. Die Idee hierbei ist, das Objekt mittels Parameter und Abhängigkeiten zu beschreiben. Diese Modelle werden entweder offline [ETR] oder online während des Trackens erzeugt [RFT][VTD]. Allerdings birgt das auch Nachteile, wie zum Beispiel bei detailreichen Hintergründen. Hier kann ein solcher Tracker leicht fehlschlagen. Deshalb wird neben dem Objekt auch die Umgebung

modelliert, zum Beispiel als negative Klasse, wobei das Objekt dann die positive wäre. Hierbei sind vor allem lernfähige Tracker von Bedeutung, da sie, im Gegensatz zu ihren statischen Pendants, während der Abarbeitung das Wissen über das Objekt und seine Umgebung sammeln, wodurch eine erhöhte Flexibilität erreicht wird. Dies geschieht mittels Erzeugnung von Klassifier [ONS][ENT][OBT].

HIER NOCH WAS ZUM SILHUETTENREPRÄSENTATION???? Neben der Modellierung lässt sich auch die Methodik spezifizieren. Hierbei wird von drei Grundformen ausgegangen: Point-Tracking, Kernel-Tracking und Silhouette-Tracking. Jede hat ihre spezifischen Eigenschaften und werden je nach unterschiedlichem Einsatz der späteren Applikation genutzt.

**Point-Tracking** Allgemein gesrpochen wird das zu verfolgende Objekt durch Punkte repräsentiert, die von einem Bild $t-1$ in $t$ gesucht werden. Da dieses Verfahren eine Reihe von Schwierigkeiten, wie das Handhaben von Verdeckungen oder falsche Punktdetektionen, aufweißt, gibt es unterschiedliche Ansätze, um ihnen zu begegnen und Lösungen zu finden. Diese werden in *deterministische* und *statitische* Methoden eingeteilt.

Deterministische Ansätze definieren eine Kostenfunktion $f$ für jedes Objekt in $t-1$ und versuchen durch Minimierung dieser Kosten und einer Reihe Bedingungen (Constraints) das korrespondierende Objekt in $t$ zu finden. Constraints können hierbei sein:

- Das Objekt verändert sein Erscheinungsbild von einem Bild zum nächsten nicht oder nur sehr wenig [FPIS]
- Die maximale Geschwindigkeit, mit der sich ein Objekt bewegt, begrennzt den Radius, in dem das Objekt im Folgebild erscheinen kann.
- Richtungs- und Geschwindigkeitsänderungen eines Objekts sind eher klein.
- Objekte in der realen Welt sind starr, weshalb die Entfernung zweier Punkte sich nicht ändert.
- ...

Diese Bedingung sind nicht auf die deterministischen Methoden beschränkt, sondern finden auch in den statitischen Ansätzen Verwendung.

Statistische Methoden modellieren Zustandräume zu bestimmten Objekteigenschaften wie zum Beispiel Geschwindigkeit oder Position im Bild. Der Tracking-Verlauf wird dann als eine Sequenz von Zuständen $X^t : t = 1, 2, \ldots$ aufgefasst, wobei, und das ist der Unterschied, das Bildrauschen $W^t : t = 1, 2, \ldots,$ das unweigerlich auftritt, in die Berechnung einfließt. So wird die Statusänderung über die Zeit für ein sich durch eine Sequenz von Bildern bewegendes Objekt beispielsweise durch die Formel $X^t = f^t(X^{t-1}) + W^t$ berechnet. Um solche Zustandsräume für das Objekttracking, also für das Schätzen der Positionsänderung des Objekts von einem Bild zum Folgebild, gibt es wiederum verschiedene Ansätze, die hier jedoch nicht näher vorgestellt werden. Zu nennen sind in diesem Zusammenhang allerdings der *Kalman*-Filter[KAF] für lineare Systeme und der *Particle*-Filter [PAF] für nichtlineare Systeme, die mit großem Erfolg in Tracking-Algorithmen verwendet wurden.

**Kernel-Tracking** Anders als beim Point Tracking dient hier eine Repräsentation des gesamten Objekts beziehungsweise einer Objektregion als Vorlage, und nicht nur einzelne Punkte. Auch hier unterscheiden sich einzelne Systeme und Ansätze hinsichtlich der Objektdarstellung, der Methodik und auch der Anzahl der zu verfolgenden Objekte. Zum Beispiel wird das Objekt mittels Template, oftmals bestimmte Color Features oder Histogramme, repräsentiert, das dann im aktuellen Bild gesucht wird. Die Postition wird dann mittels Ähnlichkeitsberechnung, beispielsweise durch Kreuzkorrelation, ermittelt. Am Ende wird die Region des Bildes als Objekt markiert, die dem Template am ähnlichsten ist. Der Nachteil dieses *Template-Matching*-Verfahrens liegt vor allem in der rechenintensiven Methodik, da sie im Ansatz brute-force ist - es wird das gesamte Bild nach dem Template abgesucht. Dabei werden auch unterschiedliche Skalierungen, Orientierungen oder Transformationen des Templates gesucht, wodurch der eigentliche Berechnungsaufwand erzeugt wird. Verbesserungen in der Laufzeit können beispielsweise durch die Einschränkung des Suchraums oder die Arbeit mit integralen Bildern[INT] erreicht werden.

Ein weiteres Verfahren, das auch in TLD Verwendung findet, ist das Tracken mittels *optischem Fluss*, die es durch Lucas und Kanade[OPT] vorgeschlagen und später durch Tomasi und Kanade[LKT] erstmals zum Tracken implementiert wurde. Die Grundidee ist die Berechnung eines Vectorfeldes, unter Annahme konstanter Helligkeit, das für jeden Pixel im Bild die Bewegungsrichtung und -geschwindigkeit

von einem Bild zum nächsten enthält. Für das Tracken wird nun ein ROI im Bild (das Objekt) ausgewählt. Dann werden aus dieser Region bestimmte Pixel für die Berechnung der Position des Objekts mittels des optischen Flusses im Folgebild herangezogen. Im Anschluss wird die Qualität der Berechnung mittels affiner Transformation zwischen dem gesuchten und dem gefundenen Patch evaluiert. Wenn die mittlere quadratische Abweichung aller Pixel zu hoch ist, wird das Ergebnis verworfen, ansonsten wird das Tracken forgesetzt.

Die genannten Verfahren eignen sich vor allem zum Single-Object-Tracking. Für das gleichzeitige Verfolgen von mehreren Objekten gibt es andere Kernel-Tracking-Algorithmen, die hier allerdings nicht näher vorgestellt werden.

**Silhouette-Tracking** Nicht jedes Objekt, das getrackt werden soll, kann durch eine einfache geometrische Figur wie ein Rechteck oder ein Elypse repräsentiert werden. Deshalb wird mittels Silhoutte-Tracking versucht, solche komplexen Objekte anhand ihrer Kontour zu repräsentieren. Außerdem motiviert zu diesem Ansatz die Tatsache, dass auch der Mensch Objekte nur anhand ihrer Silhouette erkennen kann. Da in dieser Arbeit dieses Verfahren keine Rolle spielt, sei es nur zur Vollständigkeit erwähnt (ODER SOLLTE ICH DOCH NOCH MEHR DAZU SCHREIBEN?)

Welche Methode letzendlich genutzt wird, hängt von dem späteren Einsatzgebiet des Trackers ab.

Dabei gehen die meisten Tracking-Algorithmen z.B. von einer gleichförmigen Bewegung des Objekts aus, also ohne plötzliche Bewegungsänderung. Die Geschwindigkeit beziehungsweise die Beschleunigung bleiben konstant, wodurch vorherige Informationen über das Objekt, wie Position und Größe, für das Tracken genutzt werden können und somit das Problem vereinfachen [OTS].

Allerdings haben die meisten Tracker eins gemein: Sie versagen, wenn das Objekt kurzzeitig nicht mehr sichtbar ist, entweder wenn es den Bildbereich verlässt oder zu stark durch ein anderes Objekt verdeckt wird. Zusätzlich neigen sie meist zum Drift. Sie verlieren im Laufe der Abarbeitungszeit ihren Fokus und müssen dann reinitialisiert werden.

## 2.2 Object Detection

Object Detection ist das Finden eines Objekts in einem Bild beziehungsweise einer Bildsequenz. Grundlage bildet hier vor allem die Repräsentation des Objekts im

Speicher und die damit verbundene Suchlogik. Im Laufe der letzten Jahre wurden auch hier unterschiedliche Repräsentationen entwickelt und untersucht [OTS]:

**Punkte** oder auch nur ein Punkt, der Centroid, repräsentieren das Objekt, was besonders für kleine Objekte in einem Bild nützlich ist.

**Rechtecke** oder Ellipsen, also zweidimensionale geometrische Figuren bilden das Objekt und bilden Grundlage vor allem für die Berechnung von Transformationen des gesuchten Objekts, um entsprechende Änderungen zu modellieren.

**Silhouetten** und Konturen bilden die Begrenzung des Objekts, und werden vor allem bei der Detektion von komplexen Objekten genutzt.

**Patches** des Objekts, die als Templates für die Suche dienen.

**Multi-Shape-Patches** bilden Teile eines zusammenhängenden Objekts. Die Verbindungen können durch unterschiedliche Verfahren, wie z.B. kinetic motion models, berechnet werden.

Alle genannten und ungenannte Repräsentationsverfahren können und werden in der Praxis und je nach Anwendungsgebiert kombiniert werden. Die große Herausforderung besteht nun vor allem im Segmentieren der gewählten Repräsentationen. Je nach Anwendungsfall ist eine solche Berechnung technisch leicht oder kompliziert zu realisieren. Einige Verfahren werden nun kurz vorgestellt.

**Point-Detectors** bla

**Background Substraction** bla

**Image Segmentation** bla

## 2.3 Maschinelles Lernen

Seit Mitte der 1990er Jahren halten Methoden aus dem Bereich des Machine Learning Einzug in das Gebiet der Objekterkennung. Und heutzutage gibt keinen neuen Detection-Ansatz mehr, der ohnen einen gewissen Teil *Intelligenz* auskommt, wie

auch TLD. Im Folgenden wird daher ein kurzer Überblick über verschiedene Lern-Ansätze gegeben mit Verweisen auf Implementierungen in der Bildverarbeitung gegeben.

### 2.3.1 Methoden

Eine genaue Definition für Machinellen Lernen zu finden ist schwierig, da die Einsatzgebiete der verwendeten Methoden und Ansätze sehr vielfältig sind. Eine der erste und sicher treffensten ist die Definition von Arthur Samuel (1995):

> Maschine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.

Eine aktuelle und eher mathematische Formulierung findet Tom Mitchel (1998):

> Well-posed Learning Problem: A computer program is sait to *learn* from experience $E$ with respect to some task $T$ and some performance measure $P$, if its performance on $T$, as measured by $P$, improves with experience $E$.

Im Falle der Objektdetektion ist "Finde das Objekt" die Aufgabe $T$, die zugrundeliegenden Beispiele, also die Daten, die das Objekt beschreiben, sind die Erfahrung $E$ und der Erfolg "Objekt gefunden" ist $P$.

Lernalgorithmen lassen sich in verschiedene Klassen enteilen:

- Superviced Learning,
- Unsupervised Learning,
- Reinforcement Learning und
- Recommender Systems,

wobei die ersten beiden Klassen im Bereich vermehrt im Bereich der Bildverarbeitung Verwendung finden.

Ohne im Detail Maschinelles Lernen vorzustellen, besteht jede Methode im Wesentlichen aus drei Komponenten: Das Training-Set, also das Model $M$ mit den $m$ Beispielen, das als Wissensbasis dient, der Lernalgorithmus (Kostenfunktion??), der als Eingabe $M$ erwartet, und der Bewertungsfuktion $h$, die für eine Eingabe $x$ eine Schätzung $y$ vornimmt. Die Funktion $h$ bestimmt hierbei die Abhängigkeit der Daten in $M$.

**Supervised Learning** Überwachtes Lernen. Man gibt Beispiele mit einem entsprechenden Ergebnis (Wert oder Klassefizierung) vor. Man sagt auch, dass die Beispiele gelabelt sind. Der Algorithmus kann dann anhand dieser gelernte Beispiele bestimmen, wie das neue Beispiel bewertet werden muss (also entweder wird ein Wert bestimmt oder eine Klassifzierung vorgenommen.

Hierbei unterscheidet man zwei Problemarten, die sich hinsichtlich der Berechnungsart und ihres Outputs unterscheiden:

1. Regression und
2. Klassifikation.

Regession wird verwendet, wenn zu einer bestimmten Eingabe ein konkreter (reeller) Wert erwartet wird. Klassifikation versucht hingegen die Eingabe einem diskreten Wert, also z.B. einer Klasse, zuzuordnen. Für Details wird an dieser Stelle auf die Fachliteratur verwiesen.

**Unsuperviced Learning** Hier versucht der Algorithmus für eine Anfangsmenge an Beispielen eigene Labels zu finden. Dazu werden sie anhand bestimmter Kriterien zu bestimmten Klassen zugeordnet (z.B. Entfernung). Ein Beispiel ist z.B. das Clustern von Daten.

## 2.4 Beispiele für Tracking-Implementationen

Das Feld der Objekterkennung ist weit und es gibt eine Vielzahl von unterschiedlichen Anwendungsszenarien. Im Folgenden wird eine kleine Auswahl solcher Szenarien vorgestellt, welche die oben genannten algorithmischen Ansätze implementieren.

**Gesichtserkennung** durch Farbe, Bewegung, Kombination aus beiden (alles eher schlecht), Besser: Neuronal Netze, Modelbasiert und das allerbester: Weak Classifier von viola and jones,

**Fußgänger** background seperation, foreground-silhuette-extraction kopf-detection, template-models

**Straßenschilder** bla

**Schrift** bla

**Autos** bla

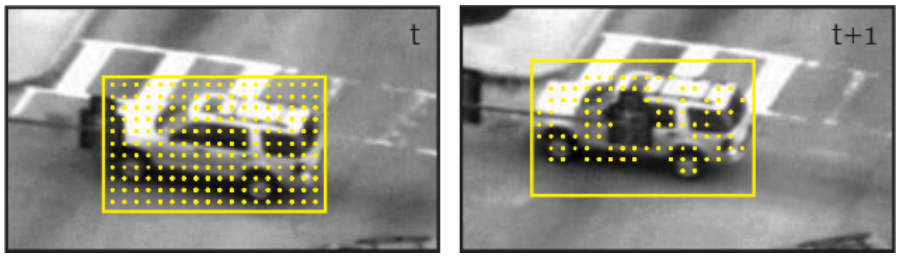


Figure 2: Arbeitsprinzip des Median Flow Trackers. Alle Punkte, die sowohl in $t$ als auch in $t+1$ vorhanden sind, werden als Grundlage für die Berechnung der BoundingBox gentutzt. (Quelle: [MFT])

# 3 Tracking Learning Detection - TLD

KURZE ZUSAMMENFASSUNG DES KAPITELS

## 3.1 Tracking

Die Tracker-Komponente hat im Wesentlichen die Aufgabe die Bewegung eines Objekts $o$ zu schätzen, wobei typischerweise die Annahme besteht, dass das Objekt in einer Bildsequenz sichtbar ist. Es gibt in der Praxis eine Reihe von Ansätzen, die sich hinsichtlich der Repräsentation des Objekts, zum Beispiel als Punkte, Kontouren, Modelle oder optischer Fluss, als auch der algorithmischen Verarbeitung unterscheiden.

In dieser Arbeit wird, wie im grundlegenden TLD-Ansatz, der Media Flow Tracker[MFT] implentiert. Der Grundgedanken basiert auf der *forward-backward consistency assumption*, die besagt, dass erfolgreiches Verfolgen unabhängig von der Richtung des Zeitflusses sein sollte. Als erstes wird eine Menge von gleichverteilten Punkten, definiert durch die BoundingBox, in einem Bild $t$ erzeugt. Im nächsten Schritt erfolgt die *forward* Bestimmung des optischen Flusses mittels Lucas-Kanade-Algorithmus[OPT] im Bild $t+1$. Die Punktemenge, die dadurch ermittelt wurde, wird nun *backward* für die Bestimmung des optischen Flusses im Bild $t$ genutzt. Die Informationen aus beiden Lucas-Kanade-Berechnungen werden zur Bestimmung es *Forward-Backward Errors (FB)* um fehlerhafte Punkte zu filtern. Die verbliebenen Punkte bestimmen die Transformation der BoundingBox. Abbildung 2 verdeutlicht das Prinzip.

### 3.1.1 Der optischen Flusses und der Lucas-Kanade-Algorithmus

Die Berechnung des optischen Flusses entspricht im Wesentlichen der folgenden Aufgabe: Finde den Punkt $x$ aus Bild $I_0$ im Bild $I_1$ beziehnugsweise in einer Sequenz von Bildern $I_1, .., I_n$. Lucas und Kanade gehen zur Berechnung von drei Annahmen aus.

1. *Brightness constancy.* Das Erscheinungsbild eines Pixel eines Objekts in einer Sequenz bleibt unverändert. Für Grauwertbilder bedeutet das, dass die Helligkeit (*brightness*) eines Pixels Konstant bleibt, wenn dieser von Bild zu Bild verfolgt wird.
$$I_0(x) = I_1(x + d)$$

2. *Temporal persistence.* Die Positionsänderung eines Objekts in einer Bildsequenz ist gering. Das bedeutet, dass die Geschwindigkeitsvectoren klein sind. Diese können im zweidimensionalen Fall mit der folgenden partiellen Ableitung berechnet werden:
$$I_x u + I_y v + I_t = 0,$$
mit $u, v$ als die gesuchten Geschwindigkeitsvectoren für die Koordinatenänderung für $x$ und $y$ im Bild $I$ und $I_t$ als zeitliche Änderung zwischen den Bildern. Allerdings ist diese Gleichung für einzelne Pixel nicht eindeutig lösbar, da durch die gesuchten Vectoren $u, v$ zwei Unbekannte existieren. Der Ergebnisraum wäre somit ein Linie und kein einzelner Punkt. Dieses Problem wird durch die dritte Annahme behandelt.

3. *Spatial coherence.* Punkte in einer Nachbarschaft gehören zur einer Oberfläche und verhalten sich gleich. Der gesuchte Punkt bildet das Zentrum eines $5 \times 5$ Pixel großen Ausschnitts. Mit Hilfe der Nachbarpunkte, deren Geschwindigkeit nach Annahme gleich ist, kann das folgende Gleichungssystem mit 25 Gleichungen erzeugt werden:
$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}.$$

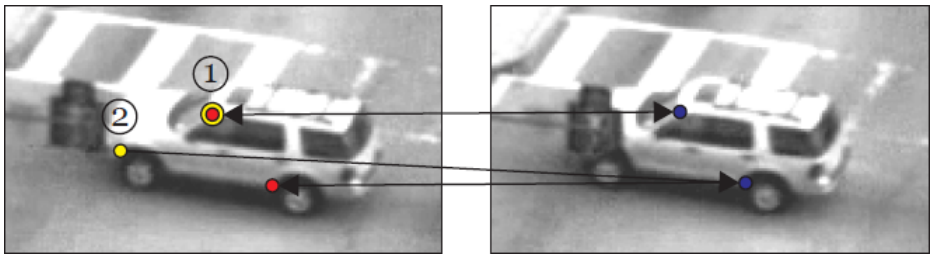


Figure 3: Erkennen von Tracking-Fehlern. Die forward-backward-Trajektorien von Punkt 1 stimmen überein, da dieser in beiden Bildern existiert. Für Punkt 2 weichen sie jedoch segnifikant ab, weil dieser im zweiten Bild verdeckt ist. (Quelle: [MFT])

Mittels Methode der kleines Quadrate kann die Lösung des Gleichungssystems minimiert. Für weitere Details zur Implementierung in *OpenCV* siehe[OCV].

### 3.1.2 FB, NCC und Median Flow Tracker

Sei $S = (I_t, I_{t+1}, .., I_{t+k})$ eine Bildsequenz und sei $x_t$ ein Punkt zur Zeit $t$. Unter Verwendung eines beliebigen Trackers wird $x$ in $k$ Bildern aus $S$ verfolgt. Die erzeugte Trajektorie ist $T_f^k = (x_t, x_{t+1}, ..., x_{t+k})$, mit $f$ für *forward* der Länge $k$. Ziel ist es nun die Gültigkeit von $T_f^k$zu bestimmen. Hierzu wird der Punkt $x_{t+k}$*backward* zum ersten Bild $I_t$ verfolgt. Der resultierende Trajektorie ist $T_b^k = (\hat{x}_t, \hat{x}_{t+1}, ..., \hat{x}_{t+k})$, mit $b$ für *backward* und $\hat{x}_{t+1} = x_{t+1}$. Der Fehler zwischen beiden Trajektorien ist definiert als $FB = distance(T_f^k, T_b^k)$, mit dem euklidischen Abstand zwischen dem initialen Punkt $x_t$ von $T_f^k$ und dem letzten Punkt $\hat{x}_t$ von $T_b^k$, also $distance(T_f^k, T_b^k) = ||x_t - \hat{x}_t||$.

Abbildung 3 verdeutlicht das Erkennen von Tracking-Fehlern. Im linken Bild ist Punkt 1 in beiden Bildern sichtbar, also sind *forward*- und *backward*-Trajektorien identisch und der Tracker kann ihn korrekt verfolgen. Punkt 2 hingegen ist im rechten Bild verdeckt, was zu einer Bestimmung eines falschen Punktes führt. In der Konsequenz unterscheiden sich *forward*- und *backward*-Trajektorie und durch die Berechnung des *Forward-Backward Errors* kann dieser Fehler leicht erkannt und behandelt werden.

Neben dem Forward-Backward Error wird von Kalal et al.[TLD]ein weiteres Messverfahren implementiert, der die Leistungsfähigkeit des Trackers signifikant verbessert[MFT]. Mittel hierfür ist der *Normalized Crorrelation Cefficient (NCC)*. Es werden nicht nur die gesuchten Pixel, sondern auch deren umgebeneNachbarn

analysiert. Dazu werden $10 \times 10$ Pixel große Patches, ausgehend vom verfolgten Punkt, generiert und mit einem ebenso erzeugten Patch des möglicherweise entsprechenden Punktes auf Ähnlichkeit untersucht. Sind sich beide ähnlich genug, dann wird der gefundene Punkt als valide angesehen.

Der implentierte *Median Flow Tracker* wendet das *Forward-BackwardError*-Prinzip in Verbindung mit der Berechnung des *NCC* entsprechend [NCC] an. In Alg.1 ist der Algorithmus skizziert. Als Eingabe dienen das letzte Bild $I$ mit der dazugehörige BoundingBox$B_I$ sowie das Folgebild $J$. Zur Berechnung der gültigen Punkte werden die Mediane des FB $median_{fb}$und des *NCC* $median_{ncc}$genutzt. Hierbei werden die Punkte verworfen, deren FB größer als $median_{fb}$ und deren Ähnlichkeitswert kleiner als $median_{ncc}$ ist. Falls $median_{fb}$ größer als ein Treshold $th_{fb}$ ist, wird das Ergebnis durch LK als falsch interprätiert und verworfen. Die restlichen Punkte und $B_I$ dienen zur Berechnug der nächsten BoundingBox $B_J$. Hierzu wird der Median der Abstände aller Punkte in $I$ und in $J$ berechnet die Änderungen als Grundlage für die Verschiebnung in x- und y-Richtung genutzt. Die Skalierung von $B_J$ ist die relative Änderung der Abstände.

**Input** $I, J, B_I$
**Output** $B_J$
Generiere Punkte $p_1, \ldots, p_n$ mittels $B_I$
**for all** $p_i$
**do**
$p'_i = LK(p_i)$
$p''_i = LK(p'_i)$
$fb_i = |p_i - p''_i|$
$ncc_i = NCC(p_i, p'_i)$
**end**
$median_{fb} = median(fb_1 \ldots fb_n)$
**if** $median_{fb} > th_{fb}$ **then**
$B_J = \emptyset$
**else**
$median_{ncc} = median(ncc_1 \ldots ncc_n)$
$q = validePoints(p, median_{fb}, median_{ncc})$
$B_J = calcBoundingBox(q, B_I)$
**end**

**Algorithmus 1 :** Tracking

## 3.2 Detection

Wenn das verfolgte Objekt den Bildbereich verlässt oder durch andere Objekte verborgen wird, kann die Tracker-Komponente nicht weiterarbeiten. Das Model zur Berechnung der Trajektorie fehlt. Aus diesem Grund ist eine Detection-Komponente erforderlich, die versucht das Objekt im Bild zu finden. Typischerweise ist diese Aufgabe recht aufwendig und die Umsetzung stark von der Art der Repräsentation des Ojekt-Modells abhängig.

In dieser Arbeit wird wie in [TLD] beschrieben, ein kaskadierender Detector implementiert. Grundlage für das Durchsuchen eines Bildes ist hierbei die *Sliding-Window*-Methode [IIM]. Hierbei wird ein Gitter definiert, das über das Bild gelegt wird. Jedes Feld des Gitters ist ein Ausschnitt (*window*) des Bildes und wird einzeln durch mehrere Klassifizierer bewertet. Sollte ein solches Teilbild am Ende der Kaskade nicht verworfen worden sein, ist es ein Kandidat für das gesuchte Objekt. Die Größe des Gitters wird durchdie initiale BoundingBox bestimmt, da die einzelnen Felder Skalierungen dieser sind.

Die Kaskade des Detectors besteht aus drei Teilen, dem Varianzfilter, dem *Ensemble-Classifier* und als letzte Instanz der *NearestNeigbour-Classifier*, die im folgenden einzeln vorgestellt werden. Dabei ist der Grundgedanke, dass ein gesuchtes Objekt komplexer als der Bildhintergrund ist. Jeder Teil hat die Aufgabe, die *windows* zu Klassifizieren und entweder zu verwerfen, da sie nicht dem gesuchten Objekt entsprechen können, oder an den nächsten Klassifizierer weiterzureichen, wenn keine eindeutig negative Aussage getroffen werden kann. Nach einer kompletten Abarbeitung des Gitters werden die übrig gebliebenen Kandidaten mittels Cluster-Verfahren gruppiert. Als Kriterium dient der Grad der Überlappung der einzelnen *windows*. Aus jedem Cluster wird abschließend eine gemittelte BoundingBox berechnet, die ein mögliches gefundenes Objekt repräsentiert. Im Gegensatz zu vielen anderen Detector-Ansätzen, benötigen diese Implementation keine offline-Trainingsdaten. Die Lernkomponente, bestehend aus Ensemble-Classifier und Nearest-Neighbour-Classifier, wird mit der ersten BoundingBox initialisiert. Alle weiteren Trainingsbeispiele werden mit Hilfe des Trackers und der P-N Learning-Komponente erzeugt. Genaueres folgt im Punkt Maschinelles Lernen.

### 3.2.1 Objekt-Modell

Die Modellierung des Objekts und dessen Umgebung erfolgt in einer speziellen Datenstruktur

$$M = \{p_1^+, p_2^+, \ldots, p_m^+, p_1^-, p_2^-, \ldots, p_n^-\}.$$

Die Patches $p^+$bilden hierbei die positiven Beispiele, die das Objekt identifizieren ($p_1^+$bildet hierbei das initiale Objekt, das durch die Definition der BoundingBox erzeugt wurde). Die $p^-$ sind eine Sammlung negativer Beispiele, die aus der Umgebung der BoundingBox ermittelt werden. Während der Ausführung des Algorithmus' wird das Objekt-Modell stetig mit neuen positiven und negativen Beispielen erweitert. Die Strategie wird im Kapitel Maschinelles Lernen näher beschrieben.

### 3.2.2 Sliding-Window

Aus der initialen *BoundingBox* werden alle möglichen Skalierungen und Verschiebungen dieser Box generiert. Damit wird das gesamte Bild mit den verschiedenen Größen und Positionen, die das Objekt annehmen kann, überdeckt. Für die Erzeugung des so entstehenden Gitters werden die folgenden Parameter genutzt: Skalierungs-Schritt= 1.2, horizontale Verschiebung = 10% der Breite, vertikale Verschiebung = 10% der Höhe, Mindestgröße einer Box = $20 \times 20$ Pixel. Für ein Bild mit einer Auflösung von $640 \times 480$ Pixel ergeben sich bis zu 200.000 *BoundingBoxes*. Die tatsächliche Anzahl ist allerdings von der initialen BoundingBox abhängig. Für Geschwindigkeitsverbesserungen, kann die Mindestgröße einer Box auf $40 \times 40$ Pixel erhöht werden, wodurch nur noch bis zu 50.000 BoundingBoxes erzeugt werden. Die Resultate des Detectors werden dadurch nicht signifikant schlechter.

### 3.2.3 Varianzfilter

Die erste Stufe des kaskadierenden Classifier bildet der Varianzfilter. Dieser vergleicht alle Patches, die durch das Grid gegeben sind, und verwirft die, deren Varianz kleiner als die Varianz des gesuchten Objekts ist. Dadurch können uniforme Regionen wie Hintergründe (Himmel, Wände etc.) schnell erkannt und verworfen werden um so den Suchraum zu verkleinern.

Die Berechnung der Varianz kann sehr schnell mittels integralen Bildern (*integral images*) erfolgen [IIM]. Dabei handelt es sich um eine spezielle Datenstruktur,

die eine einfache und schnelle Summierung von Pixelwerten beliebiger Regionen in einem Bild in $O(1)$ erlaubt. Zur Veranschaulichung siehe Abb. 4. Für eine Bild $i$ der Größe $w, h$ ist das integrale Bild $ii$ eine Struktur der Größe $w+1, h+1$, wobei die erste Zeile und die erste Spalte jeweils mit 0 gefüllt sind. Alle anderen Pixelwerte entsprechen der Summe der Pixel links und über dem Pixel, addiert mit dem eigenen Wert.

| | | |
|---|---|---|
| 4 | 3 | 1 |
| 2 | 4 | 3 |
| 8 | 10 | 4 |

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 4 | 7 | 8 |
| 0 | 6 | 19 | 30 |
| 0 | 14 | 43 | 57 |

Figure 4: Darstellung der Pixelwerte eines $3 \times 3$ Bildes (links) und des dazugehörigen integralen $4 \times 4$ Bildes (rechts).

Die Berechnung kann in linearer Zeit mittels einmaliger Iteration über das Bild durchgeführt werden.

$$ii(x,y) = \sum_{x' \leq x} \sum_{y' \leq y} i(x', y').$$

Der Vorteil der integralen Bilder liegt nun in der schnelle Berechnung von Pixelwerten innherhalb rechteckiger Teilbildern, die dann für weitere Berechnungen oder als Vergleichskriterium dienen. Die Summe der Pixel für ein Quadrat $ABCD$, siehe Abb. 5, kann mit Hilfe von vier Array-Referenzen berechnet werden. Da $D$ die Summe aller verherigen Pixel ist, müssen die Werte von $B$ und $C$ subtrahiert werden, weil sie nicht Teil von $ABCD$ sind. Da allerdings der Wert von $A$ dadurch zweimal abgezogen wurde, muss $A$ abschließend wieder dazuaddiert werden. Daraus ergibt sich die Formel

$$ABCD = D + A - C - B$$

für die Summe der Pixelwerte eines Teilbildes $ABCD$.

Neben dem integralen Bild können auch weitere Strukturen, wie das quadratische integrale Bild $ii^2$ auf ähnliche Weise berechnet werden. In der Implementierung wird das auch getan, um die Varianz durch $var(p) = ii^2(p) - i(p)^2$ zu berechnen. WELCHE VORTEILE HAT DAS GENAU??? HIER MUSS NOCH WAS ZUR ERKLÄRUNG HIN!!!

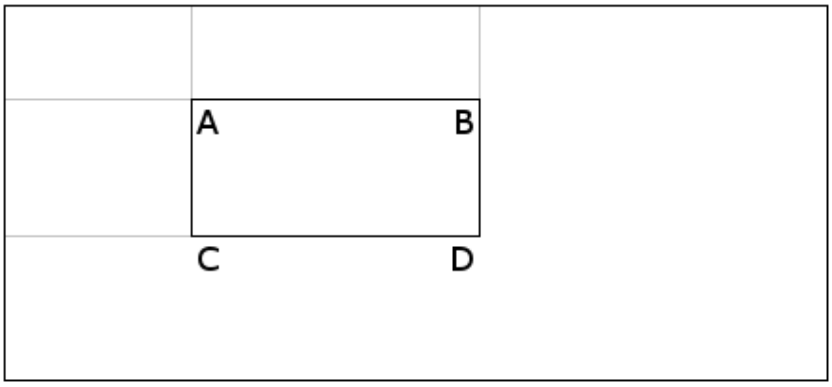


Figure 5: Teilbild $ABCD$

Auf die beschriebene Weise werden nun die Varianzen der einzelnen möglichen Objekte, die durch das Grid definiert sind, mit dem Varianzwert des aktuellen BoundingBox verglichen, und bei einem geringeren Wert verworfen. Alle übrig gebliebenen Boxen werden im folgendenen *Essemble Classifier* verarbeitet.

### 3.2.4 Ensemble-Classifier

Wurde ein Patch nicht durch den Varianzfilter verworfen, gelangt er zur zweiten Stufe der Kaskade, dem EnsembleClassifier. Dieser besteht aus einer Menge so genannter Base-Classifier, die einzeln eine Bewertung zu dem übergebenen Patch abgeben. Aus deren Ergebnisse wird ein Mittelwert gebildet, der dann die Bewertung des *Ensemble-Classifiers* ist.

**Random Ferns** Grundlage des Klassifizierers ist die in XXX vorgestellte *random fern classification*-Methode. Das Ensemble der Base-Classifier besteht aus $n$ Ferns, von denen jeder unabhängig den Patch $I$ bewertet. Dafür vollzieht jeder Fern $i$ eine Anzahl binärer Pixelvergleiche zweier zufällig gewählter Pixel im Patch, mit

$$f_i = \begin{cases} 0, & I(d_{i,1}) < I(d_{i,2}) \\ 1, & sonst \end{cases}$$

Aus den Vergleichen der einzelnen Features, also zweier Pixelwerte, wird ein

binären Code $x$ erzeugt. Der entsprechende dezimale Wert ist wiederum Index für ein Feld von A-Posteriori-Wahrscheinlichkeiten $P_i(y|x)$, mit $y \in 0, 1$. Aus allen berechneten Wahrscheinlichkeiten jedes Ferns wird dann ein Mittelwert berechnet und falls diese kleiner als 0.5 ist, wird der Patch verworfen.

Die zufällig gewählten Features werden bei der Initialisierung des *Ensemble-Classifiers* berechnet und bestehen während der Laufzeit des Programms unverändert. Da der Vergleich zweier Pixel ein sehr schwaches Unterscheidungskriterium ist, muss eine große Anzahl von Features für jeden Fern erzeugt werden. Anders ausgedrückt sollte das Feld der A-Posteriori-Wahrscheinlichkeiten möglichst viele Einträge haben. Für ein $P_i$ ergibt sich $2^d$ als Feldgröße, wobei $d$ die Anzahl der Features für ein $i$ist. In der Implementierung führt jeder Fern 13 Pixelvergleiche durch, was in 8.192 möglichen Codes resultiert. Die A-Posteriori-Wahrscheinlichkeit wird durch $P_i(y|x) = \frac{\#p}{\#p+\#n}$ berechnet, mit $\#p$ als Anzahl der positiven und $\#n$ als Anzahl der negativen Bewertungen eines Ferns $i$ für den Code $x$ ist.

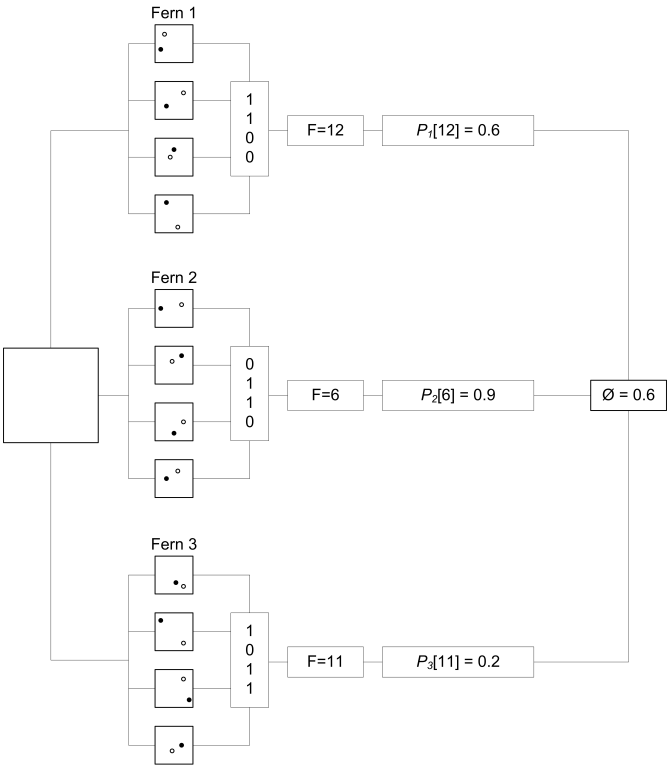


Figure 6: Prinzipielle Arbeitsweise der *Base-Classifier*.

Abbildung 6 verdeutlich das Prinzip. Das Ensemble besteht aus drei Ferns mit jeweils vier Feature. Die in jedem Kästchen dargestellte Punkte stellen die Pixel im Bild da und werden jeweils paarweise verglichen. Pro Kästchen also der Pixelwert des schwarzer mit dem des weißen Punktes. Die vier binären Vergleiche pro Fern resultiert in einem binären Code, für Fern 1 ist dieser beispielsweise 1100. Dieser Code wird nun dezimal interprätiert und dient als Index im *Posterior*-Array, im genannten Beispiel also $1100_2 = 12_{10}$ und damit $P_1[12] = 0.6$.

### 3.2.5 Template-Matching und Nearest-Neighbour-Classifier

**Template-Matching** Die letzte Stufe des Klassifizierers ist ein Nearest-Neighbour-Classifier, der mittels Template-Matching die verbliebenen Patches klassifiziert. Bei diesem Verfahren wird der gesuchte Patch (das Template) $T$ direkt in einem Bild $I$ gesucht, indem es über dieses geschoben und mittels einer Matching-Methode, in dieser Implementation durch Berechnung des (normalisierten) Korrelations-Koeffizienten, verglichen wird. Dabei wird der relative Mittelwert von $T$ mit dem relativen Mittelwert von $I$ verglichen. Das Ergebnis ist ein Wert zwischen 1(perfekter Match) und $-1$(perfekter Missmatch). 0 bedeutet, dass zwischen $T$ und $I$ keine Korrelation exisitert. Da es sich hierbei um eine sehr konservative und rechenintensive Vergleichsmethode handelt, bildet sie den Abschluss der Kaskade.

Der Korrelations-Koeffizient wird wie folgt berechnet:

$$CC(x,y) = \sum_{x',y'} [T'(x',y') \times I'(x+x',y+y')]^2$$

$$T'(x',y') = T(x',y') - \frac{1}{(w \times h) \sum_{x'',y''} T(x'',y'')}$$

$$I'(x+x',y+y') = I(x+x',y+y') - \frac{1}{(w \times h) \sum_{x'',y''} I(x+x'',y+y'')}$$

Um Fehler zu reduzieren, die durch eine unterschiedliche Beleutung des Templates $T$ und des Bildes $I$ entstehen können und damit das Ergebnis verfälschen, wir der Koeffizient normalisiert. Zusätzlich wird das Ergebnis umgerechnet, das Ergebnis im Intervall $[0..1]$ liegt.

$$S(x,y) = 0.5(NCC(x,y)+1)$$

$$NCC(x,y) = \sqrt{\sum_{x',y'} T(x'y')^2 \times \sum_{x',y'} I(x+x', y+y')^2}.$$

**Nearest-Neighbour-Classifier** Positive und negative Beispiele, die dem Klassifizierer für den Vergleich dienen, sind genormte $15 \times 15$ Pixel große Patches und werden durch das Objekt-Modell $M$ repräsentiert. Der zu überprüfende Patch wird mit allen positiven und negativen Beispielen verglichen und es wird jeweils der größte Korrelations-Koeffizient berechnet. Diese dienen wiederum als Grundlage für die Ähnlichkeitsberechnung des Patches zur positiven Klasse. Es wird hierbei die relative Ähnlichkeit $S^r(p,M) = \frac{S^+}{S^+ + S^-}$ berechnet und mit einem Threshold-Wert $\Theta$ verglichen, mit $\Theta = 0.65$[1]. Gilt $S^r(p,M) > \Theta$, wird der Patch als positiv klassifiziert.

Alle Patches, die zu diesem Zeitpunkt nicht durch die Kaskade verworfen wurden, bilden die Ausgabe der Detektor-Komponente. Typischerweise sind es mehrere Patches, die sich sehr ähneln und stark überlappen, und im Allgemeinen Teile des gesuchten Objekts repräsentieren. Das ist $\Theta$ geschuldet, denn die Patches werden nicht mit einem klaren Label für "ist Objekt/ist kein Objekt" versehen. Aus diesem Grund werden mittels Cluster-Verfahren Gruppierungen gebildet, aus denen abschließend die BoundingBox für das mögliche Objekt generiert wird.

### 3.2.6 Cluster-Verfahren

Nachdem der Detector eine Reihe von Patchen klassifiziert und mit $p^+$ gelabelt hat, ergibt sich ein Problem. Im Idealfall sollten alle Patches, die postitiv sind, also als Objekt erkannt wurden, mit 1 und alle anderen mit 0 bewertet werden [BAB]. Dies ist in diesem Verfahren jedoch nicht möglich, da die Bewertung des Nearest-Neighbours Werte zwischen $[\Theta..1]$ generiert und eine eindeutige Klassifizierung nicht möglich ist. Wegen des Sliding-Window-Ansatzes wird der Detector fast immer Kandidaten finden, die sich sehr nahe an der eigentlichen Detektion befinden, siehe Abbildung [].

[1] Werte für $\Theta$ zwischen $0.5 - 0.7$ sind ebenfalls zulässig und bringen ähnliche Resultate [TLD].

## 3.3 Maschinelles Lernen

Die Tracking- und die Detection-Komponenten arbeiten unabhängig von einander, weshalb deren Ergebnisse zusammengeführt und bewertet werden müssen. Dies geschieht durch P/N-Learning [PNL]. Gleichzeitig bildet dieser Ansatz die Grundlage des Lernens und damit des Trainierens des Fern- und des NearestNeighbour-Classifiers.

Hintergrund des semi-supervised learning-Prozesses sind postive (P) und negative (N) Restriktionen (Constraints) für das Markieren (labeling) des unmarkierten Datensatzes. Im Gegensatz zu anderen Ansätzen [17,4 vom PN-L-PAPER!!!!] werden die Beispiele des unmarkierten Datensatzes, im Folgenden *Struktur*, als zusammenhängend beziehungsweise von einander abhängig angesehen. Im Fall der Object Detection bilden alle möglichen Patches des Bildes diese Struktur und die Aufgabe besteht darin jeden dieser Patches entweder als positiv, und somit dem Objekt, oder als negativ, und damit dem Hintergrund, zuzuordnen. Da ein Objekt in einer Videosequenz temporär nur an einem Ort sichtbar sein kann, werden die Patches in diesem Bereich eine ähnliche postiven Markierung erhalten, Patches, die weiter weg vom Objekt sind, eine negative.

Abbildung 7 gibt einen Überblick der beteiligten Komponenten und deren Zusammenhang von P-N Learning. Im Klassifizierer (i) (im TLD-Ansatz sind das Fern- und NearstNeigbour Classifier) werden die unbestimmten Daten aus $X_u$ gelabelt. (ii) bilden die Constraints zur Bewertung der gelabelten Patches, ändern gegenballs die Labels und erweitern die Trainingsmenge (iii), die wiederum für das Training (iv) des Klassifizierers genutzt wird. Die genaue Erläutering der Arbeitsweisen erfolgt im Anschluss.

### 3.3.1 P-N Learning

Sei $X$ ein Beispielraum (Feature-Space) und $x \in X$ ein Beispiel, und sei $Y = \{-1, 1\}$ ein Labelraum und $y \in Y$ ein Label. Eine Menge von Beispielen $X$ und mit entsprechenden Labels $Y$ sei die markierte (gelabelte) Menge $(X, Y)$. Die Aufgabe von P-N Learning ist die Funktion $f : X \rightarrow Y$, parametrisiert mit $\Theta$, von einer initialen Menge $(X_l, Y_l)$ zu lernen und mittels ungelabelter Daten $X_u$ dessen Arbeitsweise zu verbessern. . $f$ ist somit ein Classifier, der mittels bootstrapping trainiert wird.

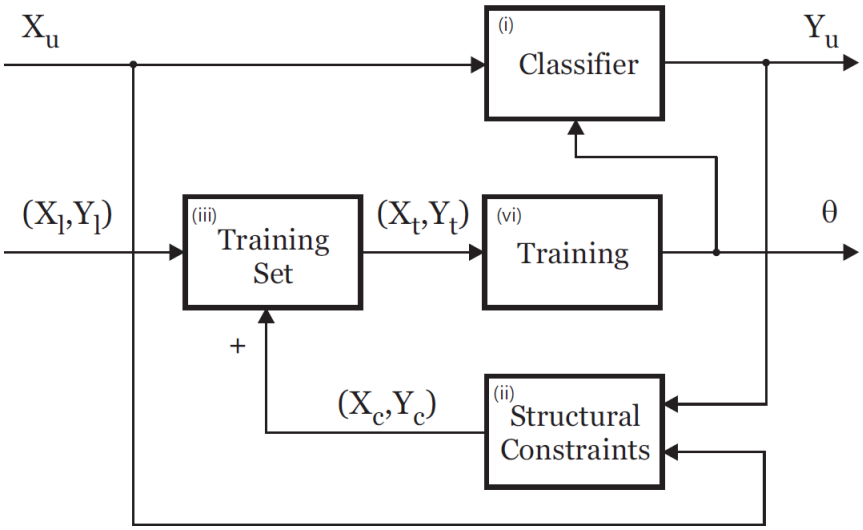


Figure 7: Arbeitsweise P-N Learning (Quelle: [PNL])

**Bootstrapping** Lernen das Klassifizierers durch Erweiterung des Trainingsets mittels Beispielen, die durch die Constraints aus den ungelabelten Daten gewonnen werden. Initialisierung durch bereits gelabelte (also positive und negative Beispiele). Von da an erfolgt die Abarbeitung iterativ.

In Schritt $k$ hat der Klassifizierer bereits $k-1$ mal Labels zu unmarkierten Beispielen bestimmt, mit $y_u^k = f(x_y|\Theta^{k-1})$, wobei in jedem Schritt nur ein Beispiel berechnet wird. Die Constraints bewerten im Anschluss die erzeugte Ausgabe und prüfen, ob die Labels den erwarteten Bedingungen genügen. Sollte dies nicht der Fall sein, werden sie gegenfalls umgelabelt und werden der Trainingsmenge hinzugefügt. Ist dieser Schritt abgeschlossen, wird der Klassifizierer mit Hilfe der erweiterten Trainingsdaten trainiert und die Abarbeitung beginnt mit der nächsten Iteration $k+1$ von vorn.

**Constraints** Die Constraints dienen zum Bewerten der Detection-Ergebnisse und zur Generierung der Trainingsmenge. Die wichtigste Aufgabe besteht darin, nur solche Beispiele für die Lernprozedur zu filtern, die "gut genug" und gleichzeitig "nicht zu gut" sind. Dazu wird das Ergebnis des Trackers mit der Ergebnismenge des Detectors verglichen. Hat der Detector ein Objekt gefunden, das mit dem des Trackers übereinstimmt, wird nicht gelernt, da die verwendete Datenbasis des Clas-

sifiers anscheinend gut genug war. Findet der Tracker allerdings ein Objekt, das nicht mit der Ergebnismenge übereinstimmt, wird geschätzt, welches Ergebnis am plausibelsten ist. Dies geschieht auf Grundlage der vorhanden Datenbasis.

Hier können zwei Fälle eintreten:

1. Das Ergebnis des Trackers ist "besser": In diesem Fall ist die gefundene Version des Objekts noch nicht in der Datenbasis und dient als Grundlage für ein positives Beispiel in der nächsten Lernphase.

2. Das Ergebnis des Detectors ist "besser": Damit wurde das gesuchte Objekt gefunden und der Tracker hat anscheinend aus unterschiedlichen Gründen den Fokus verloren. Deshalb muss er reinitialisiert werden, und zwar auf das Objekt, das der Detector gefunden hat.

Ein Constraint ist in erster Linie eine Funktion, die als Parameter eine Menge von gelabelten Beispielen des Classifiers $(X_u, Y_u^k)$ erwartet und eine Menge von Beispielen $(X_c^k, Y_c^k)$, deren Label geändert wurden, als Ergebnis liefert. Die Anzahl dieser Constraints kann in diesem Fall beliebig sein, allerdings werden sie in zwei Kategorien geordnet: $P$ und $N$.

*P-constraints* dienen zur Identifikation von Beispielen, die vom Classifier zwar als negativ bewertet, von den Constraints allerdings als positiv angesehen werden. *N-constraints* hingegen identifizieren negative Beispiele, die vom Classifier fälschlicherweise als postitiv bewerten wurden. Dabei sind in der $k$-ten Itereration $n^+(k)$ die Anzahl der Beispiele, die ein neues postives Label und $n^-(k)$, die Anzahl der Beispiele, die durch die Constraints ein neues negatives Label erhalten haben. Dadurch werden zum einen neue postivie Beispiele erzeugt, die der Classifier noch nicht kennt und zusätzlich werden negative Beispiel generiert, die in der nächsten Iteration nicht fälschlicherweise als positiv bewertet werden.

## 3.4 Zusammenspiel der Komponenten

Nachdem die einzelnen Teile des Tracking-Detection-Learning-Systems im Detail vorgestellt und behandelt wurden, erfolgt in diesem Abschnitt eine kurze Erläuterung des Zusammenspiels aller Komponenten.

Prinzipiell ist für die Initalisierung von TLD eine Nutzereingabe erforderlich. Diese Dient einzig der Markierung des gesuchten Objekts. Entweder wird in einem

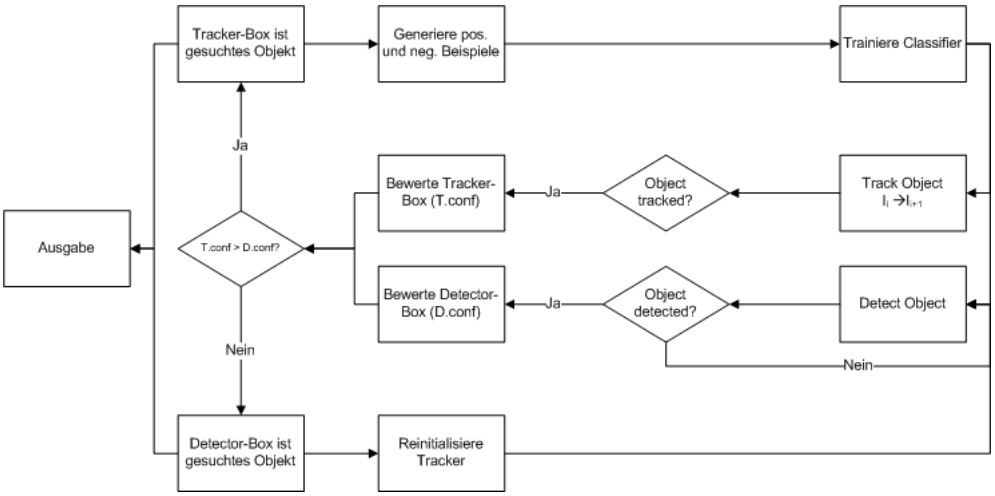


Figure 8: Zusammenspiel der Komponenten im Programmablaufplan

Ausgabebild das Objekt durch ein Rechteck markiert, oder ein bereits erzeugtes Model wird geladen, wordurch die Classifier mit den entsprechenden Datenbasen initialisiert werden. Ein dritte Möglichkeit biete eine vorgeschaltete Komponente, wie zum Beispiel ein alternativer Klassifizierer, der für die Erkennung eines bestimmten Objekts trainiert wird und die initialie BoundingBox liefert. Nachdem die Initialisierung durchgeführt wurde, arbeitet TLD selbständig und ohne weitere Eingaben von Außen. Die Tracking- sowie die Detection-Komponente arbeiten vollkommen unabhängig und parallel. Beide liefern nach der Abarbeitung ein Ergebnis in Form einer Box, die Kandidaten für das gesuchte Objekt sind. Aufgrund des Sliding-Window-Ansatzes und des abschließenden Clusterings ist es möglich, dass der Detector mehr als eine mögliche Box als Ausgabe hat, also kein eindeutiges Ergebnis liefert. In diesem Fall wird für die folgende Bewertung nur die Box des Trackers berücksichtigt, und die Ergebnismenge des Detectors wird verworfen.

Liefern Detector und Tracker jeweils eine Box, wird mittels NearestNeigbour-Classifier bestimmt, welche der beiden Boxen am Ehesten dem gesuchten Objekt entspricht.

$T.conf \geq D.conf$ Die Box des Trackers hat einen höheren Confidense-Value als die des Detectors. Das bedeutet, dass die Classifier, die Teil des Detectors sind, das Objekt in seiner gefundenen Form nicht erkannt habe, was wiederum bedeutet, dass

neue Beispiele generiert werden müssen. Der Trainingsprozess wird angestoßen, die synthetischen positiven Beispiele werden generiert und der Lernfunktion übergeben. Diese bewertet wiederum jedes Beispiel mittels des NearsestNeighbour-Classifier und wenn es "zu gut" ist, das heißt $conf \geq .65$, wird das Beispiel verworfen.

$T.conf < D.conf$ Der Detector liefert das bessere Ergebnis, was wiederum bedeutet, dass der Tracker das Objekt nicht finden konnte, weil es verdeckt oder teilweise nicht sichtbar war, oder der Tracker gedrifftet ist. In jedem Fall wird der Tracker mit der Detector-Box re-initalisiert. Ein Training findet in diesem Fall nicht statt.

Der Tracker dient in erster Linie zur Erzeugung neuer positive Lernbeispiele,wenn sein Ergebnis das des Detektors übertrifft. Hierzu wird der Conf-Wert (???) der Tracker-Box mit dem Conf-Wert

## 3.5 Initialisierung

Hier die Erläuterung zu Initialisierung der BoundingBox. Aufgrund der Probleme muss ein Model vorher gelernt und verfeinert werden (HIER TESTEN).

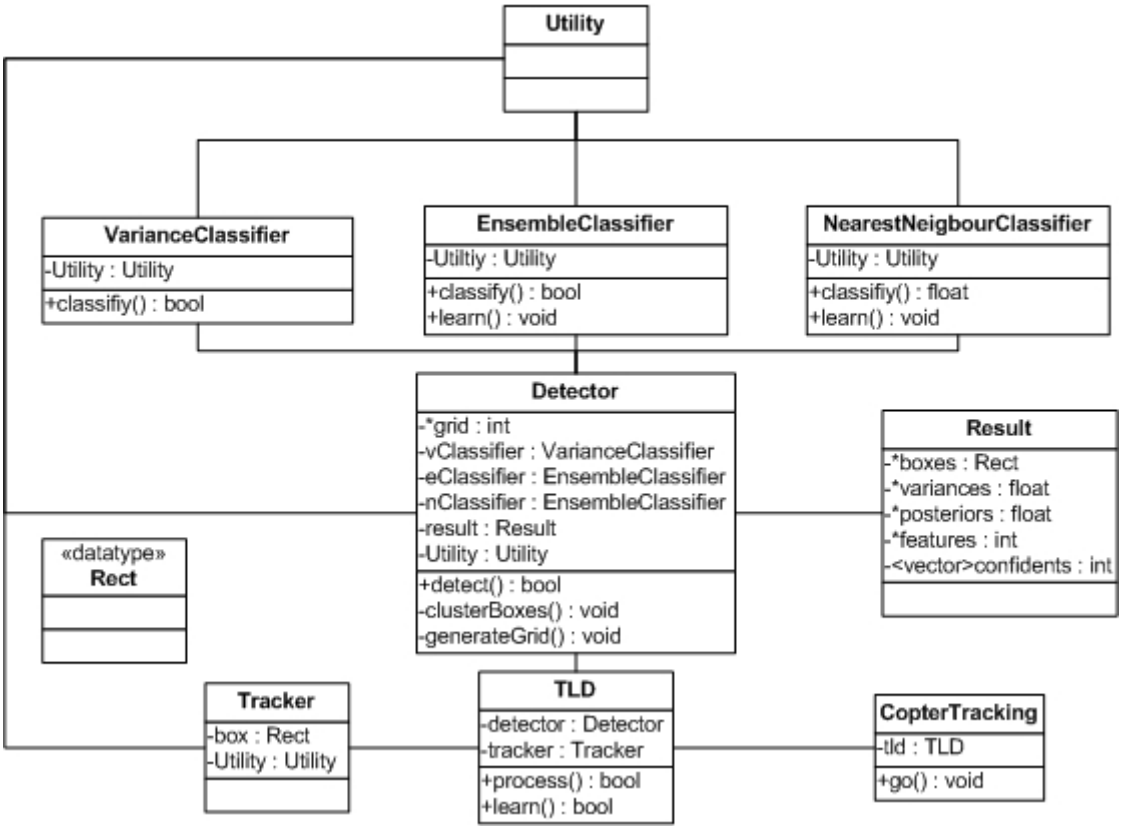


Figure 9: Klassendiagramm zur Implementation mit den wichtigsten Attributen und Methoden

# 4 Implementierung und Testszenarien

Beschreibung des Kapitels

## 4.1 Programmierung

Die Implementierung erfolgte in C++. Zusätzlich wurden OpenCV als unterstützendes Framework genutzt. Die Grundlegende Klassenstruktur ist in Abbildung 9 verdeutlicht.

### 4.1.1 Klassenstruktur

**Klasse *CopterTracking*** Sie bildet den Einstieg ins Programm. Mittels OpenCV wird das Kamera- beziehnungsweise Videobild Frame für Frame eingelesen. Außerdem erfolgt hier die Definition des Objekts und die Übergabe des nächsten zu analysierenden Bildes sowie der Objektrepräsentation durch die aktuelle Box an die Klasse *TLD*.

**Klasse *TLD*** Diese Klasse bildet das Kernstück des Algorithmus und implementiert alle Methoden für das Zusammenspiel der Komponenten. Kernstück bildet die Funktion *process()*, die als Eingabe aktuelle Bild $I$ mit der zuletzt gefundenen BoundingBox $b$ erwartet und die BoundingBox $b'$ berechnet, sofern das Objekt gefunden wird. Neben der Initialisierung aller Komponenten beim Start des Programms, implementiert sie auch die Auswertung der Ergebnismengen von Tracker und Detector in *compareBoundingBoxes()*, die Beide unabhängig von einander die Daten verarbeiten. Das ist die Implementierung von P-N Learning (3.3).

**Klasse *Tracker*** Die Implementierung des MedianFlow-Trackers ist in dieser Klasse realisiert.

**Klasse *Detector*** Diese Klasse implementiert die Kaskade aus den drei Classifiern *VarianceClassifier*, *EnsembleClassifier* und *NearestNeigbourClassifier*. Weiterhin hält der Detector das Model des gesuchten Kopters als *Result*-Objekt vor und berechnet in der Funktion *clusterBoxes()* die Cluster, aus denen dann ein oder mehrere finale Boxen berechnet werden. Initial definiert der Detector mittels der Funktion *generateGrid()* das *Grid* als eine **int**-Zeigervariable, die Grundlage des sliding-window-Ansatzes ist. Es hat sich während der Implementierung gezeigt, dass eine Indexierung auf einer Zeigervariable die Abarbeitungsgeschwindigkeit gegenüber einer Implementierung des Grids in Form eines Vector-Objekts stark erhöht.

**Klasse *VarianceClassifier*** In einer Schleife werden nacheinander alle durch das *Grid* gegebene Windows mittels Klassifizierer-Kaskade bewertet. Die erste Stufe, bildet dies Klasse VarianceClassifier. Die Methode *classify()* zeichnet sich für die Bewertung verantwortlich und füllt die entsprechenden Varianz-Werte des *Result*-Objekts.

**Klasse *EnsembleClassifier*** Stufe zwei der Kaskade ist durch die Klasse *EnsembleClassifier* definiert, die im Detail den vorgestellten *FernClassifier* repräsentiert. Auch diese hat eine *classify()*-Methode, vefügt jedoch zusätzlich über eine *learn()*-Methode, über die der Klassifizierer in der Lernphase trainiert wird.

**Klasse *NearestNeigbourClassifier*** Die letzte Stufe ist der *NearestNeigbourClassifier* mit den zur Klassifizierung und zum Trainieren benötigten Methoden *classify()* und *learn()*. Neben der Nutzung im Detector dient er auch zur Bewertung der Tracker- und Detector-Box durch die *P-N Learning*-Komponente in der *TLD*-Klasse für die Lernphase.

**Klasse *Result*** Während der Bewertung jeder einzelnen Box durch die Kaskade der drei Classifier, werden neben der eigentlichen Klassifizierung, auch die ermittelten Werte (variance, posteriors, confidence) und die durch ein Rechteck defininierte Box in einem Result-Objekt zwischengespeichert. Die berechneten Werte werden in Zeiger- beziehnugsweise Vector-Variablen gespeichert. Vor allem erstere bieten einen sehr schnellen Speicherzugriff und damit eine laufzeitstarke Abarbeitung von TLD.

**Klasse *Utility*** Diese Klasse liefert eine Reihe Klassenmethoden für unterschiedliche Berechnungen, die an verschiedenen Stelle im Programm genutzt werden.

**Zusätzliche Angaben** In den ersten Versionen des Programms wurde ein streng objektorientierter Ansatz verfolgt. Das bedeutet, dass das Objekt-Modell auch als eine solche Klasse definiert war. Und alle Objekte bildeten wiederum eine Grid, das als Vector-Variable hinterlegt war. Wie sich allerdings gezeigt hat, war die Abarbeitungsgeschwindigkeit bei einem großen Grid und damit einer Menge von Objekten nicht zufriedenstellend. Trotz Übergabe des Grids als call-of-reference-Zeigervariable war nur mit einer kleinen Gridgröße von maximal 4000 Objekten eine passable Abarbeitung möglich. Dies entsprach allerdings bei weitem nicht der in der originalen Matlab-Implementierung beschriebenen ca. 50.000 Boxen. Indexierungen über Zeigervariablen sind um ein Vielfaches schneller und deshalb wurde ein Refactoring durchgeführt, in dem alle laufzeitkritischen Vector-Variablen durch Zeiger ersetzt wurden. Die Herausforderung bestand im Anschluss darin,

eine möglichst geschickte Indezierung zu erzeugen, das zum Teil mehrdimensionale Werte in einer eindimensionalen Variable hinterlegt werden müssen.

## 4.2 Tests

Die Ergebnisse der originalen Implementation von TLD mittels Matlab sind beeindruckend...(HIER NOCH ETWAS ZU DEN ERGEBNISSEN). Aus diesem Grund viel die Wahl auf TLD.

Nach der Implementation sollte die Eignung von TLD auf das Koptertracking getestet werden. Dazu wurden verschiedene Testszenarien erstellt und durchgespielt, die nun kurz vorgestellt werden.

Als Datenquelle dienten in erster Linie Video- und Kamerasequenzen, die mittels einer handelsüblichen SRL in einer Auflösung von $640 \times 480$ Pixel aufgenommen wurden. Als Modell wurde der *MINI-QUADROCOPTER QG 550 XS* gewählt (BILD). Es sollten vor allem zwei wesentliche Dinge überprüft werden:

1. Unter welchen Umständen bezwiehungsweise welche Kriterien müssen erfüllt werden, damit ein Kopter erfolgreich gefunden und verfolgt werden kann.

2. Kann durch spezielle Lernphasen und damit einer Verfeinerung des Models der Erfolg des Trackens verbessert werden.

### 4.2.1 Videosequenzen

Die verwendeten Videosequenzen decken Szenarien ab, in denen ein Kopter getrackt werden muss. Es wurden verschiedene mögliche negative Einflussfaktoren wie Änderungen der Lichtverhältnisse, wechselnde Bewegungsgeschwindigekeiten und Bewegungsrichtungen mit Innen- und Außenaufnahmen simuliert. Neben der manuellen Initialisierung von TLD durch den Nutzer, dienten die in den Szenen erzeugten Models in einigen Testläufen ebenfalls als Initialisierung und wurden zusätlich erweitert.

Da TLD bereits durch verschiedene, zum Download zur Verfügung stehende Evaluierungssequenzen getestet worden ist, und es für den speziellen Fall des Kopter-Trackings keine Daten vorhanden waren, wurden ausschließlich eigene Sequenzen verwendet.

### 4.2.2 Kamerasequenzen

Da Videosequenzen nur bedingt für die Verfeinerung eines Models geeignet sind, wurden zuätlich Kamerasequenzen genutzt. Dadurch kann auf Tracking-Detection-Probleme und der Verfälschung des Models durch fehlerhafte Fokussierung von TLD reagiert werden.

# 5 Auswertung und Ausblick

Object Detection ist eines der herausfordernsten Forschungsfelder in Computer Vision. Obwohl die Entwicklung von robusten und gut funktionierenden Ansätzen in den letzten Jahren stark zugenommen hat und unter kontrollierten Bedingungen auch sehr gute Ergebnisse liefert, ist das Problem des Suchen und Finden von einem oder mehreren Objekten in der "realen" Welt weiterhin ungelöst [ODS].

## 5.1 Evaluation

Die Evaluierung von TLD erfolgt mit Hilfe eine Standardverfahrens zur Beurteilung von Klassifikatoren und findet auch in [TLD]. Hierbei dienen die quantitativen Ergebnisse des Klassifizierers in jedem Testszenario als Maß. Da TLD ein binärer Klassifizier ist, gibt es zwei Arten von Fehlern die auftreten können. Entweder wird das Objekt in die Klasse $A$ eingeordnet, obwohl es zu $B$ gehört, oder es wird als Teil von $B$ bestimmt, gehört allerdings zur Klasse $A$. $A$ und $B$ sind in Falle von TLD "gefunden" und "nicht gefunden".

Als Vergleichsbasis dient der Ground Truth. Das ist der Bereich, der die genaue Position des gesuchten Kopters im Bild markiert, gegeben durch eine BoundingBox $BB_{GT}$. Dieser wird mit dem Ergebnis von TLD verglichen, der ebenfalls eine BoundingBox $BB_{TLD}$ bei erfolgreicher Detection beziehnungsweise erfolgreichem Tracking liefert. Zwischen beiden Boxen wird die Überlappung bestimmt und geprüft, ob sie größer oder kleiner eines Thresholds $\omega$ ist. Daraus ergeben sich vier Fälle:

**True-Positive** $t_p$ Die Überlappung ist größer als $\omega$.

**True-Negative** $t_n$ Es wurde kein Objekt gefunden und es wurde auch keines erwartet.

**False-Positive** $f_p$ Der Algorithmus findet ein Objekt, obwohl keines erwartet wurde, oder die Überlappung ist kleiner als $\omega$.

**False-Negative** $f_n$ Der Algorithmus findet kein Objekt, obwohl eines erwartet wurde, oder die Überlappung ist kleiner als $\omega$.

In jedem Testdurchlauf werden die Anzahl $t_p$, $t_n$, $f_p$ und $f_n$ gezählt und mittels statistischer Gütekriterien ausgewertet.

Wie auch bei [TLD] sind diese Kriterien Genauigkeit (Precision) $P$, Trefferquote(Recall) $R$ und das gewichtet harmonische Mittel $F$.

HIER WEITER!!!

$P$ ist hierbei die Anzahl wahr-positiver Detections geteilt durch die Anzahl aller Detections, $R$ ist die Anzahl wahr-positiver Detections geteilt durch die Anzahl der Detections, die erwartet waren und $F$ kombiniert die beiden Messungen in der Form $F = 2 \times PR/(P + R)$ (Was heißt das eigentlich?).

TLD beeindruckt durch sehr gute Ergebnisse beim Tracken von sehr unterschiedlichen Objekten, wie Fußgänger, Autos und Gesichtern.

Der Ansatz ist nur bedingt geeignet. Probleme:

- Speichern der Beispiele in Form von $15 \times 15$ patches, weil zu viel Hintergrund gespeichert wird,
- Objekt (Kopter) hat filigrane Struktur,
- "Veschmelzung" des Objekts mit dem Hintergrund

Weiteres...

- Ein paar Worte zur Umsetzung (openCV)
- Erweiterungsmöglichkeiten - z.B. Verfolgung mehrerer Kopter gleichzeitig, Optimierungsvorschläge, weitere Einsatzmöglichkeiten (da TLD an sich "alles" tracken kann)
- Zusammenfassung
- ohne zusätzlich Detectionsverfahren (eventuell Tiefenbilder oder andere nicht-visuelle Ansätze) kaum möglich
- Umwelteinflüsse in der Natur sind zu stark
- Einfluss der unterschiedlichen Beleuchtungen der Szene
- Hintergrund (Bäume etc)
- Problem bei zusätzlichen Verfahren: Rechenleistung des Roboters(?)

Verbesserungen/Ansätze

- eventuell stärke Orientierung an natürlich Seh-prozessen bzw. Sensoren in der Natur (Auge der Fliege)
- Eventuell ein anderer Ansatz: Vielleicht sollte man eine 3-D Repräsentation des Objekts während des Trackens erzeugen? Der Mensch macht es sicher auch nicht anders. Schließlich weiß man ja oft, wonach man suchen muss, auch wenn das Erscheinungsbild des Objekts nicht mit dem bereits "Erlernten" übereinstimmt...
- Articulated shape models > hierbei werden Teile des Objekts einzeln repräsentiert und zu einem ganzen Zusammengesetzt.
- Eventuell nicht so allgemeine Ansätze, sondern vorher viel mehr Wissen über das Objekt, den Kopter, als Grundlage nehmen. Eventuell ein 3- -Modell erzeugen und aus dem ermittelten Bild alle entsprechende Objekte auf dieses "mappen".

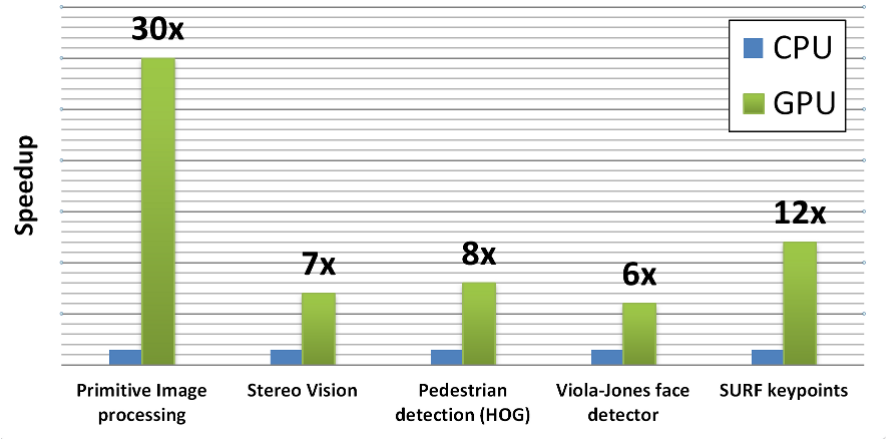


Figure 10: Tesla C2050 versus Core i5-760 2.8Ghz, SSE, TBB [OCW]

# 6 Anhang

## OpenCV

Die *Open Source Computer Vision Library* OpenCV ist einen Bibliothek mit einer Sammlung von Klassen und Algorithmen in den Bereichen des machinellen Sehens, der Bildverarbeitung und des maschinellen Lernens implementiert und zur (meist) freien Verfügung stellt. Wurde anfangs nur die Sprache C unterstützt, gibt es heute ein Reihe von Implementierungen in C++, Python, Java und auch MATLAB. Außerdem werden sowohl die gängigen Betriebssysteme Windows, Linux und Mac OS, als auch das mobile OS Android unterstützt.

Zur Zeit befinden sich spezielle Module in der Entwicklung (basierend zum Beispiel auf CUDA), die es erlauben, Algorithmen expliziet auf der GPU (*Graphic Processor Unit*), also auf dem Grafikprozessor, auszuführen. Die Auslagerung auf die für die Verarbeitung von Bilddaten optimierte Hardware hat vor allem Geschwindigkeits Vorteile 10

## OpenTLD

Im Laufe der Bearbeitung des Themas wurde eine C++-Implementierung von TLD. OpenTLD adaptiert die originale Matlab-Version, erweitert allerdings die Detector-Kaskade um einen weiteren Filter, den Foreground-Filter. Dieser dient in erster

Linie zur Reduzierung der möglichen BoundingBoxes.

Mit Hilfe eines Forground-Filters kann ein sich bewegendes Objekt mittels Background-Substraction aus einer Videosequenz separiert werden[QUELLE]. Dieses Verfahren wird vor allem bei statischen Kameras verwendet, die fest montiert. In einer sehr dynamischen Anwendung, wie das Finden und Verfolgen von sich bewegenden Objekten mittels auf einem Flugroboter montierten Kamera unter Nichtlaborbedingungen, hat eine solche Komponente somit keine Relevanz auf die Robustheit des Algortihmus. Außerdem ist dieser Filter nicht Teil der originalen Veröffentlichung von Z. Kalal. Aus diesen Gründen wurde auf eine Implementierung verzichtet. Ansonsten wurden Teil von OpenTLD zu Analysezwecke und standen zur Vervollständigung der eigenen Implementierung Pate.

Darstellung der verwendeten C++-Klassen. Erläuterungen zu den Lernalgorithmen (Ferns, RandomForest, NearestNeighbour etc.)

# List of Figures

## References

[TLD] Z. Kalal, k. Mikolajczyk, and J. Matas. Tracking-Learning-Detection. *Pattern Analysis and Machine Intelligence*, IEEE Transactions on (Volume:34 , Issue: 7 ), pages 1409-1422, July 2012

[MFT] Z. Kalal, k. Mikolajczyk, and J. Matas. Forward-Backward Error: Automatic Detection of Tracking Failures. *Proc. 20th Int'l Conf. Pattern Recognition*, pages 23-26, 2010.

[OPT] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. *International Joint Conference on Artificial Intelligence*, pages 674-679, 1981.

[OCV] G. Bradski and A. Kaehler. Learning OpenCV: Computer Vision with the OpenCV Library. *O'Reilly Media, 1st edition*, Oct. 2008.

[NCC] J.P. Lewis. Fast normalized cross-crorrelation. *Vision Interface. Canadian Image Processing and Pattern Recognition Society*, pages 120-123, 1995.

[IIM] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–511–I–518, Los Alamitos, CA, USA, Apr. 2001

[VID] Video Analyse - Therie and practice.

[PYR] Jean-Yves Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker : Description of the algorithm, *Intel Corporation – Microprocessor Research Labs*, 2002.

[BAB] M. B. Blaschko. Branch and Bound Strategies for Non-maximal Suppression in Object Detection, volume 6819 of Lecture Notes in Computer Science, chapter 28, pages 385– 398. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[OTS] A. Yilmaz, O. Javed, and M. Shah, Object tracking: A survey. *ACM Computing Surveys (CSUR)*, vol. 38, no. 4, 2006.

[PNL] Z. Kalal, J. Matas, and K. Mikolajczyk. P-N learning: Bootstrapping binary classifiers by structural constraints. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 49–56. IEEE, June 2010

[ODS] D.K.Prasad. Survey of The Problem of Object Detection In Real Images. *In International Journal of Image Processing (IJIP)*, Volume (6) : Issue (6), pages 441 - 466, 2012

[STV] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," International Joint Conference on Artificial Intelligence, vol. 81, pp. 674–679, 1981.

[GFT] J. Shi and C. Tomasi, "Good features to track," Conference on Computer Vision and Pattern Recognition, 1994.

[KBT] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-Based Object Tracking," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 5, pp. 564–577, 2003.

[TUP] I. Matthews, T. Ishikawa, and S. Baker, "The Template Update Problem," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 6, pp. 810–815, 2004

[SMAT] N. Dowson and R. Bowden, "Simultaneous Modeling and Tracking (SMAT) of Feature Sets," Conference on Computer Vision and Pattern Recognition, 2005.

[RDT] A. Rahimi, L. P. Morency, and T. Darrell, "Reducing drift in differential tracking," Computer Vision and Image Understanding, vol. 109, no. 2, pp. 97–111, 2008.

[ROAM] A. D. Jepson, D. J. Fleet, and T. F. El-Maraghi, "Robust Online Appearance Models for Visual Tracking," IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1296–1311, 2003.

[RFT] A. Adam, E. Rivlin, and I. Shimshoni, "Robust Fragments-based Tracking using the Integral Histogram," Conference on Computer Vision and Pattern Recognition, pp. 798–805, 2006.

[ETR] M. J. Black and A. D. Jepson, “Eigentracking: Robust matching and tracking of articulated objects using a view-based representation,” International Journal of Computer Vision, vol. 26, no. 1, pp. 63–84, 1998.

[RVT] D. Ross, J. Lim, R. Lin, and M. Yang, “Incremental Learning for Robust Visual Tracking,” International Journal of Computer Vision, vol. 77, pp. 125–141, Aug. 2007.

[VTD] J. Kwon and K. M. Lee, “Visual Tracking Decomposition,” Conference on Computer Vision and Pattern Recognition, 2010.

[ONS] R. Collins, Y. Liu, and M. Leordeanu, “Online Selection of Discriminative Tracking Features,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 10, pp. 1631–1643, 2005.

[ENT] S. Avidan, “Ensemble Tracking,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 2, pp. 261–271, 2007.

[OBT] H. Grabner and H. Bischof, “On-line boosting and vision,” Conference on Computer Vision and Pattern Recognition, 2006.

[MIP] Milan Sonka and J. Michael Fitzpatrick, Handbook of Medical Imaging : Medical Image Processing and Analysis Volume 2

[LCD] Tseng, D.-C., I-Ling Chung, Pei-Lin Tsai, and Chang-Min Chou, "Defect classification for LCD color filters using neural network-based decision tree classifier," International Journal of Innovative Computing, Information and Control, vol.7, no.7 (A), pp.3695-3707, 2011.

[PED] R. Benenson, M. Mathias, R. Timofte, and L. V. Gool, “Pedestrian detection at 100 frames per second,” in Proc. IEEE Int’l Conf. on Computer Vision and Pattern Recognition , 2012, pp. 2903–2910.

[OCW] http://opencv.org/platforms/cuda.html, 29.09.2013

[FPIS] SETHI, I. AND JAIN, R. Finding trajectories of feature points in a monocular image sequence. IEEE Trans. Patt. Analy. Mach. Intell. 9, 1, 56–73, 1987

[KAF] Kalman, R. E., A New Approach to Linear Filtering and Prediction Problems., Transactions of the ASME–Journal of Basic Engineering, Vol 82, Series D, pp. 35 - 45, 1960

[PAF] Gordon N.J., Salmond D.J. and Smith A.F.M., Novel approach to nonlinear/non-Gaussian Bayesian state estimation. IEE-Proceedings-F, 140 , 107 - 113, 1993

[INT] H., BELL, J. W., AND WU, F. 2002. Very fast template matching. In European Conference on Computer Vision (ECCV). 358–372.

[LKT] Carlo Tomasi and Takeo Kanade. Detection and Tracking of Point Features. Carnegie Mellon University Technical Report CMU-CS-91-132, April 1991.

## Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Diplomarbeit in diesem Studiengebiet erstmalig einzureichen.

Berlin, den November 20, 2013 .....

## Statement of authorship

I declare that I completed this thesis on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this nor a similar work has been presented to an examination committee.

Berlin, November 20, 2013 .....