HTW Berlin
Fachbereich 4
Angewandte Informatik

Seminararbeit Ki und Robotik WiSe 2018-9

Ein-und-Ausgabemöglichkeiten eines einfachen Supervised-Learning Systems

Vorgelegt von

Florian Sauer s0552569

Camilo Ocampo s0546867

Angefertigt unter der Leitung von Patrick Baumann

Berlin, 7 Februar 2019

Inhaltsverzeichnis

Einleitung

Unsere Seminararbeit ist eine Erweiterung der im Kurs vorgestellten Exercise 1. Sie erweitert diese in drei Punkten. Daten werden in Echtzeit von einer Kamera eingelesen, ausgewertet und vorhergesagt und anschließend als Text to Speech ausgegeben. Die Bilddaten werden über ein Peer-to-Peer-Netzwerk an das ROS-Netzwerk weitergeleitet. Das Projekt dient als "Proof of Concept" für ein "Image to Speech"-System. Diese Dokumentation dient dazu, die genutzten Machine-Learning-Aspekte und Algorithmische Prinzipien hinter dem Projekt zu erläutern, die verwendeten Technologien zu vergleichen, die aufgetretenen Probleme aufzulisten und die Implementierung zu beschreiben.

Publisher-Subscriber-Principle

Der Publisher verbindet sich per TCP mit einem zentralen Broker, welcher im Fall von ROS im ROSCORE eingebaut ist. Der Publisher informiert den Broker, dass er eine Nachricht N unter einer Topic T veröffentlichen möchte. Der Broker stellt nun die nötige Infrastruktur bereit, um die Nachricht an mehrere Subscriber der Topic weiterzuleiten. Wenn ein neuer Subscriber sich an der Topic anmeldet, empfängt dieser die letzte veröffentlichte Nachricht. Das Publish-Subscriber Schema arbeitet asynchron. Subscriber müssen deshalb z.B. Callbacks einsetzen, um die empfangenen Nachrichten zu verarbeiten. Callbacks sind Funktionen, welche per Referenz an eine andere Funktion übergeben werden, damit diese die Callback-Funktion zu einem späteren beliebigen Zeitpunkt aufrufen kann. Dieser Callback wird aufgerufen, wenn eine

neue Nachricht vom Broker empfangen wird (Wiki:ROS/Tutorials). Die für dieses Projekt genutzte Klasse “RosSubscriber“ fungiert als Wrapper-Klasse für die eigentliche Subscriber-Klasse des rospy-Moduls. In dieser Klasse muss lediglich die “handle”-Methode implementiert bzw überladen werden. Diese wird intern dann als Callback an den eigentlichen rospy-Subscriber übergeben.

Supervised Learning

Sowohl bei der Exercise-1 als auch bei der Belegarbeit wurde Supervised Learning eingesetzt. Der Kern von Supervised Learning basiert darauf, ein sogenanntes Model anhand von Trainingsdaten zu trainieren, welches dann unbekannte bzw. zukünftige Datensätze vorhersagt und auswertet. Das Model wird so eingerichtet, dass es die gewünschten bzw. richtigen Labels den zugehörigen Objekten zuordnet. Der Grund, warum der Ansatz “Supervised Learning” heißt, ist, dass ein Mensch die Labels schon kennt bzw. kontrolliert und inwiefern diese zu den vorliegenden Objekten passen (vgl. Raschka seite 3-4). Weiterhin handelt es sich bei diesem Beleg um ein sogenanntes “multi class classification problem”. Wir haben diskrete “Class labels” , in unserem Fall die Zahlen Null bis Neun, welche wir auf Bilder anwenden wollen um sie somit zu identifizieren.

Verwendete Netzwerktechniken des Versuchsaufbaus

Der Versuchsaufbau wurde mittels einer virtuellen Maschine realisiert, verwendet wurde VM Virtualbox von Oracle. Das Host-System war ein Windows-10-Rechner, das Gast-System ein

Ubuntu-System. Um Daten vom Host-System an das Gast-System per TCP-Sockets zu transportieren, wurde sowohl ein “Host Only Adapter”, als auch ein “NAT-Adapter” getestet und verwendet. Letzterer birgt das Problem, dass das Gast-System zwar eine Verbindung zum Host-System aufnehmen kann, nicht jedoch umgekehrt. Um das NAT zu durchdringen, wurde ein Peer-to-Peer-System verwendet, welches eine “NAT-Durchdringungs-Funktion” besitzt. Dieses P2P-System wurde außerhalb des Kurses privat entwickelt. Einfach dargestellt, wird das NAT über einen öffentlich zugänglichen dritten P2P-Knoten durchdrungen (siehe Abbildung 1). Für bessere Latenzen ist der “Host Only Adapter” zu empfehlen. Alternativ kann das Host-System einen SSH-Server anbieten, zu dem sich ein SSH-Client aus dem Gast-System verbindet und einen “Reverse-Tunnel” aufbaut.

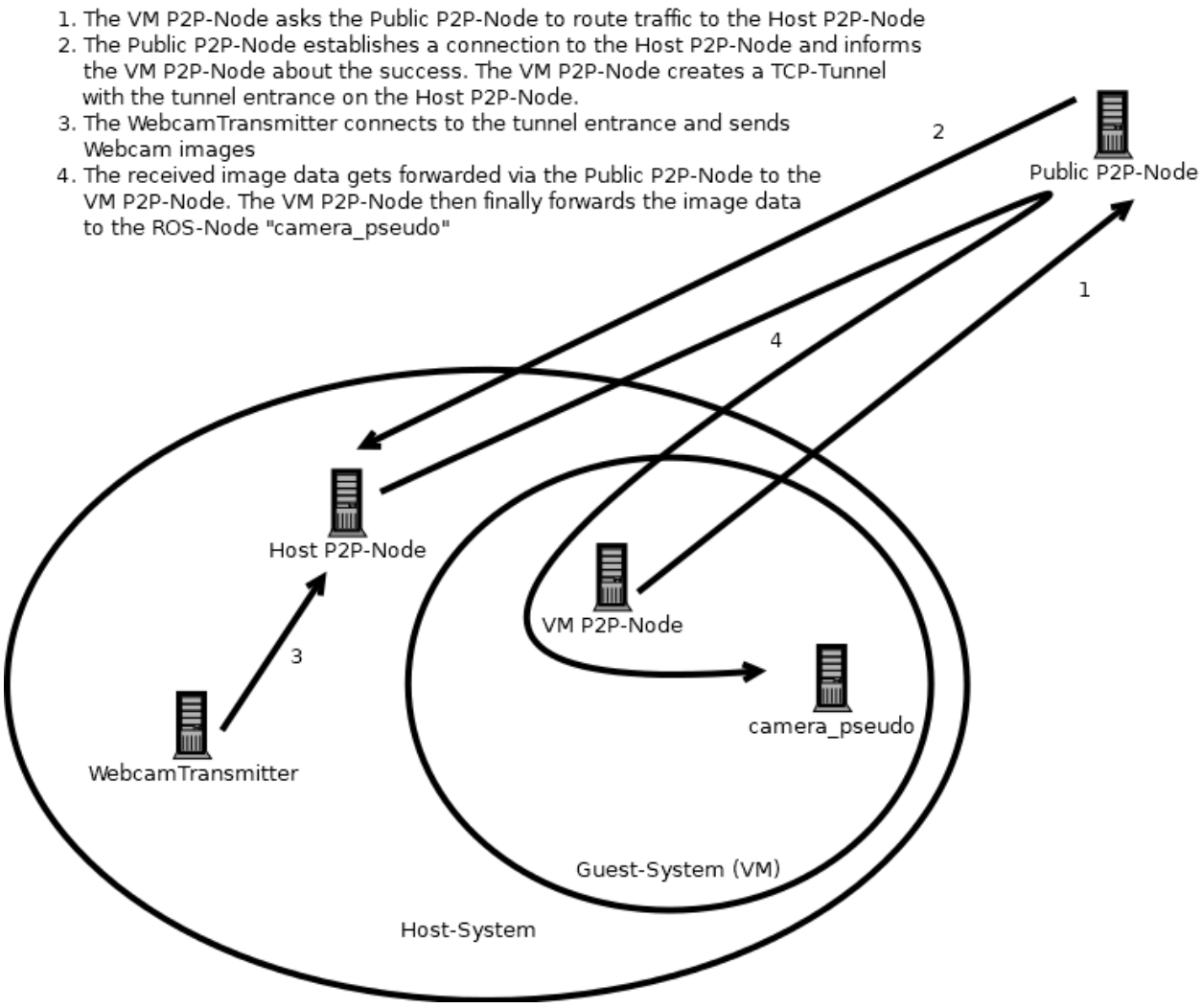


Abbildung 1: Routing, NAT Durchdringung und Datenübermittlung mittels P2P-Netzwerk

Vergleich zwischen Peer-to-Peer-Prinzipien und ROS-Prinzipien

Ein Vergleich bzw. die Einordnung von ROS im Peer-to-Peer-Kontext ist kaum möglich.

ROS setzt mehr auf einen zentralen Verwaltungsknoten, zu dem sich mehrere Client-Knoten verbinden und anmelden. Dies ist insofern notwendig, da Knoten zu verschiedenen Topics Daten publishen und subscriben können (siehe Publisher-Subscriber-Prinzipien). Diese Daten müssen

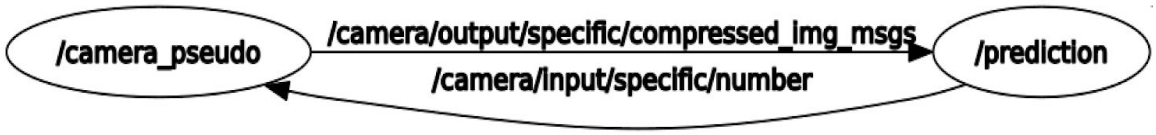
zentral verwaltet werden. Die Kommunikation läuft also primär über den zentralen Verwaltungsknoten (ROSCore). Peer-to-Peer-Prinzipien setzen jedoch auf einen dezentralen Ansatz, bei dem es zwar auch zentrale Knoten oder Verwaltungsserver geben kann, diese jedoch keine zentrale Kommunikationsrolle spielen und nur Verzeichnis- oder Registrierungsdienste anbieten. Der eigentliche Datenaustausch wird immer zwischen den Knoten selbst abgewickelt. (vgl. Schade, Seite 5-12)

Beschreibung der Implementierung

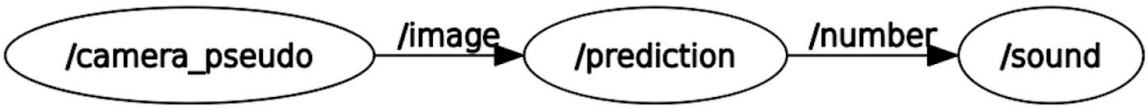
Der Beleg basiert zu Teilen auf Exercise 1. Es gibt jedoch unterschiede in der Funktionsweise. Vorhersagen werden nun nicht mehr an den Node “camera_pseudo” zurückgeschickt, sondern an den Node “sound”. Bilddaten werden ebenfalls nur noch über eine einzige Topic vom “camera_pseudo”-Node an den “prediction”-Node geschickt. Ebenfalls werden die Bilddaten nicht mehr (zufällig) aus einem bestehenden Datensatz gelesen, sondern in Echtzeit von einer Webcam aufgenommen. Da sich das Aufnehmen der Bilddaten von einer Webcam aus als schwierig herausgestellt hat, vor allem aus einer virtuellen Maschine heraus, wurden die Bilddaten vom Host-System aus per TCP-Stream an einen Server geschickt, der vom “camera_pseudo”-Node aus geöffnet wurde. Konkret ist hier die Klasse RemoteWebcamReader zu nennen. Diese besitzt ein ähnliches Interface wie die Klasse VideoCapture des cv2-Moduls. Wie in der MD-Datei schon erwähnt, waren die Hauptänderungen im Programm, die Nutzung einer Kamera in Echtzeit und eine hörbare Ausgabe der Vorhersagen. Die Ausgabe wurde nach den ROS-Prinzipien mit einem neuen Node namens “sound” gelöst. Dieser Node greift lediglich

auf die Ergebnisse des Prediction-Nodes zu und gibt sie aus. Dazu wurde die Bibliothek “Pygame” verwendet. Was die Kommunikation im ROS-Netzwerk angeht sind bis auf zwei Topics alle aus Einfachheitsgründen weggelassen worden. Die verbleibenden Topics sind “/image” (ehemals “/camera/output/webcam/compressed_img_msgs"), welche die Daten von der Kamera an den Prediction-Node überträgt und “/number” (ehemals “/camera/input/specific/number”), welche die Vorhersagen des Prediction Nodes an den Sound-Node übergibt. Ein Vergleich der RQT Grafiken sieht folgendermaßen aus:

Exercise 1:



Beleg:



Test des Programms

Zur Untersuchung der Gesamtfunktionalität des Programms wurde ein kleiner Test geschrieben (Siehe “Test”-Branch des GIT-Repos). Der Test basiert darauf, dass ein Nutzer Bilder vom Host-System aus per Tastendruck an die VM mit den eigentlichen ROS-Nodes schicken kann. Für diesen Zweck wurde die Webcamfunktionalität geändert, sodass nicht mehr kontinuierlich Daten aufgenommen und versendet werden, sondern auf eine Tastatureingabe gewartet wird

(Siehe TestWebcamTransmitter.py). Der Test wurde mit einem Handy durchgeführt, welches die Zahlen 0 bis 9 dargestellt hat. Die ursprüngliche Idee war, dass der Nutzer eine Taste zum Versenden eines Bildes verwenden kann. Dies hat den Vorteil, dass darauf geachtet werden kann, dass das Handy in einem günstigen Winkel zur Webcam steht und genug Umgebungslicht vorhanden ist. Trotz dieser Bemühungen wurde die Erfahrung gemacht, dass der Test immer wieder unterschiedliche Ergebnisse bezüglich der Korrektheit geliefert hat und die Ergebnisse eher inkorrekt waren. Dennoch konnte man gut erkennen, dass das System nicht zufällig geraten hat, sondern dass meist ähnlich aussehende Zahlen erkannt wurden (z.B. 3 versus 8, 1 versus 7). Es traten leider viele Störfaktoren auf, die zu einem suboptimalen Testergebnis beigetragen haben. Unterschiedlich starkes Umgebungslicht, die mehrfache Umkonvertierung der Bilder bei der Weiterleitung, die Reflektion und Positionierung des Handys und die Tatsache, dass der Test mit unterschiedlichen Computern durchgeführt wurde.

Schluss

Trotz der Problematik, die beim Test erläutert wurde, ist das "Proof of Concept" des Systems vollständig. Die oben erwähnten Störfaktoren können beseitigt werden, um das System performanter und "korrekter" zu machen. Ebenfalls kann das vor-angelernte Modell weiter mit Testdaten gefüttert werden. Dies ist jedoch ein langwieriger Prozess.

Literaturverzeichnis

Schade, D. (2005). Effizientes Routing in Peer-to-Peer Netzwerken, Seite 5-12. Aufgerufen unter https://domino.mpi-inf.mpg.de/intranet/ag5/ag5publ.nsf/0/FC279B9D6374B04CC1256FDC00351C82/$file/Diplom_Schade.pdf am 06.02.2019

Raschka, S. (2015). *Python Machine Learning*, Seite 3-4. Zweietauflage Birmingham: Packt Publishing ltd.

Prasanth T. Wiki: ROS/Tutorials. Aufgerufen unter https://wiki.ros.org/ROS/Tutorials am 12.02.2019