COMENIUS UNIVERSITY IN BRATISLAVA

FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS

ADVANCED ALGORITHMS FOR SEGMENTATION OF SPACE DEBRIS ASTRONOMICAL IMAGES

Master thesis

2021 Bc. Daniel Kyselica

COMENIUS UNIVERSITY IN BRATISLAVA

FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS

ADVANCED ALGORITHMS FOR SEGMENTATION OF SPACE DEBRIS ASTRONOMICAL IMAGES

Master thesis

Study program:	Applied informatics
Branch of study:	2511 Applied informatics
Supervisor:	Mgr. Stanislav Krajčovič
Consultant:	Mgr. Jiří Šilha, PhD.

Bratislava, 2021

Bc. Daniel Kyselica

58885648

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Daniel Kyselica
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Advanced algorithms for segmentation of space debris astronomical images
Pokročilé algoritmy na segmentáciu astronomických snímok kozmického odpadu

Anotácia: Počas astronomických pozorovaní uskutočnených pomocou 70 cm ďalekohľadu určeného na pozorovanie vesmírneho odpadu sú získavané snímky konkrétnej časti nočnej oblohy. Každý pixel
takejto snímky je reprezentovaný tromi údajmi – pozíciou na horizontálnej osi x, pozíciou na vertikálnej osi y a intenzitou, ktorá nadobúda hodnoty od 0 až po 65 536. Nulová, respektíve veľmi
nízka intenzita znamená, že na danej časti nočnej oblohy, ktorá korešponduje s x a y súradnicami na snímke, nebol zaznamenaný žiaden objekt. Naopak, intenzity pohybujúce sa v rádovo tisíckach
hodnôt a vyššie indikujú prítomnosť nejakého objektu, ako napríklad hviezdy, objektov slnečnej
sústavy, vesmírneho odpadu, alebo aj šumu elektrického prúdu, pozadia oblohy a iných artefaktov.
Úlohou študenta bude zoznámiť sa s už existujúcimi algoritmami používanými na riešenie
danej problematiky, naštudovať si poskytnutú odbornú literatúru, a navrhnúť a otestovať vlastný
algoritmus.

Cieľ: Návrh a implementácia pokročilých algoritmov za účelom extrahovania pozícií kozmického odpadu z astronomických CCD snímok

Literatúra: V. Kouprianov, Distinguishing features of CCD astrometry of faint GEO objects, Advances in
Space Research, Volume 41, Issue 7, 2008, Pages 1029-1038, ISSN 0273-1177, http://dx.doi.org/10.1016/j.asr.2007.04.033.
Zharkova, V., 2007. Artificial intelligence in recognition and classification of astrophysical and
medical images (Vol. 46). Springer Science &amp; Business Media.

Vedúci:	Mgr. Stanislav Krajčovič
Konzultant:	Mgr. Jiří Šilha, PhD.
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	prof. Ing. Igor Farkaš, Dr.

Spôsob sprístupnenia elektronickej verzie práce:

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

bez obmedzenia

Dátum zadania:	15.10.2019	
Dátum schválenia:	18.10.2019	prof. RNDr. Roman Ďurikovič, PhD. garant študijného programu

....

študent vedúci práce

I hereby declare that I have written this thesis by my-self, only with help of referenced literature, under the careful supervision of my thesis advisor.

. . . .

Bratislava, 2021 Bc. Daniel Kyselica

Acknowledgement

I’m extremely grateful to my supervisor Mgr. Stanislav Krajčovič for his insightful suggestions. Thanks should also go to Mgr. Jiří Šilha, PhD. for his in valuable insight into astronomy. Special thanks to my family for their support.

Abstract

During astronomic observations, images of selected part of the sky are made by the 70cm telescope specialized on space debris tracking. Every pixel of this frame can be represented by three data points – position on the horizontal x axis, position on the vertical y axis and intensity which value can differ from 0 to 65 536. If the intensity is zero or close to zero, no object has been detected on the position corresponding to the given x and y coordinates. Contrary to that, the intensity which value is in order of thousands or higher indicates presence of an orbital or extraterrestrial object such as a star, planet, space debris, or even electromagnetic field interference, celestial plane background and other artefacts. Our task is to research literature written on the topic of object segmentation, learn about existing system used to combat the issue, design and test a tracklet building process using neural networks.

Keywords: segmentation, object tracking, processing of astronomical images

Abstrakt

Počas astronomických pozorovaní uskutočnených pomocou 70 cm ďalekohľadu určeného na pozorovanie vesmírneho odpadu sú získavané snímky konkrétnej časti nočnej oblohy. Každý pixel takejto snímky je reprezentovaný tromi údajmi – pozíciou na horizontálnej osi x, pozíciou na vertikálnej osi y a intenzitou, ktorá nadobúda hodnoty od 0 až po 65 536. Nulová, respektíve veľmi nízka intenzita znamená, že na danej časti nočnej oblohy, ktorá korešponduje s x a y súradnicami na snímke, nebol zaznamenaný žiaden objekt. Naopak, intenzity pohybujúce sa v rádovo tisíckach hodnôt a vyššie indikujú prítomnosť nejakého objektu, ako napríklad hviezdy, objektov slnečnej sústavy, vesmírneho odpadu, alebo aj šumu elektrického prúdu, pozadia oblohy a iných artefaktov. Našou úlohou bude zoznámiť sa s už existujúcimi algoritmami používanými na riešenie danej problematiky, naštudovať si poskytnutú odbornú literatúru, a navrhnúť a otestovať vlastný systém na tvorbu trackletov.

Kľúčové slová: segmentácia, sledovanie objektu, spracovanie astronomických snímok

Contents

List of Figures

Introduction

In recent decades, interest in space flights has increased rapidly. Many companies and nations send their satellites to Earth's orbit. The importance of these satellites increased significantly as they serve many different functions: navigation, data trafficking, weather forecast, space observation, etc. With this increasing activity in the orbit, the population of space debris is rising. This phenomenon endangers all satellites orbiting the Earth. In the future, if we do not find a solution to this problem, the risk of sending a probe might become too high. Regarding this issue, space agencies around the world are developing systems to monitor and collect space debris. Our University created system for processing series of images of objects orbiting Earth - Image Processing Pipeline for Space Debris. In recent years, machine learning is the best performing method for various problems. It can be used for correlating objects together into more complex structures.

1. Motivation

The rising population of space debris is a threat to satellites. Most orbital debris come from human products such as pieces of satellites, rockets, protective gloves, etc. Cataloguing these objects can help us prevent unnecessary collisions by moving satellites in advance. The existing system, developed by the Department of Astronomy and Astrophysics [1], is capable of tracking objects from series of observations. The current analytic solution might produce false positive tracklets, due to its architecture. Because of the nature of the system false positives are undesirable. Machine learning is a rising field with a big potential and using a neural network to create tracklets can remove the downsides of the current system.

1.1 Space debris

Objects orbiting the Earth which are at the end of their missions, are called **space debris**. Size of these objects varies from less than millimeter up to meters. Smaller objects are almost impossible to track, yet they make up the majority of space debris. Collisions with these fast moving objects can cause high damage to satellites and space crafts, as in the figure 1.1.

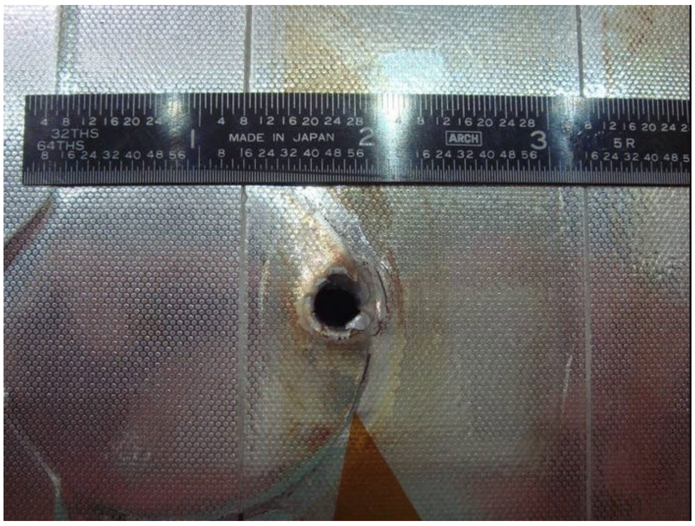


Figure 1.1: A space debris hit to space shuttle Endeavor. The entry hole is 0.25 inches wide. The exit hole is twice as large. Image downloaded from (NASA 2020).

Only objects with certain sizes at certain altitudes can be tracked by optical devices. US Space Surveillance Network (USSSN) maintains a database of tracked objects. The database consists of more than 42 thousand objects. More than a half of these objects is still orbiting the Earth. Figure 1.2 show approximate population of space debris around the Earth at low orbit (LEO) objects.

Without resolving this problem, phenomenon called *Kessler syndrome* could occur. It represents a state where rate of collisions and space debris population in orbit would increase exponentially. To prevent this scenario, we need to first track space debris and then collect it from the Earth orbit.

The information used in this section was from the master thesis - *Development of an efficient tracklet building algorithm for space debris objects* [2].

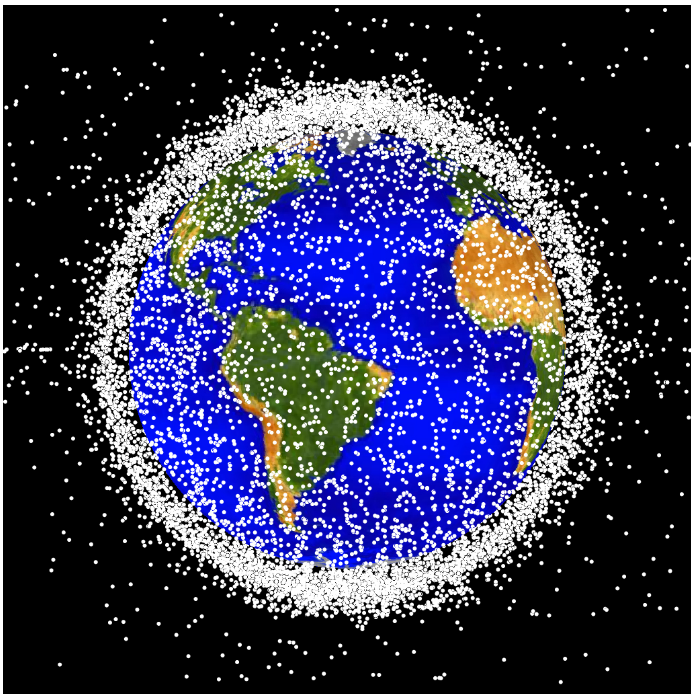


Figure 1.2: Most concentrated area for orbital debris is low Earth orbit (LEO). Image downloaded from (NASA 2020).

1.1.1 Types of orbits

Objects move around the Earth in orbits. This movement is caused by the Earth's gravity. They are divided into these many types according to ESA (The Europe space agency) article [3]:

- **Geostationary orbit (GEO)**

  Objects in GEO circle the Earth around equator at altitude around 35786 km and with speed around 3 km per second. One rotation in GEO takes exactly one Earth day which cause satellites in GEO to appear stationary over a fixed position. For example television satellites are in GEO.

- **Low Earth orbit (LEO)**

  Low Earth orbit stays for an orbit with altitude below 1000 km. Lowest LEO orbit could be only 160 km above the Earth surface. Objects travel around 7.8 km per second in this orbit. LEO objects do not have to orbit along Earth's equator. Their plane can be tilted. LEO is the most used orbit around the Earth.

- **Medium Earth orbit (MEO)**
  MEO include wide range of orbits between LEO and GEO. Like in LEO, objects in GEO do not have to travel on specific path around the Earth. These orbits are commonly used for navigation satellites like the European Galileo system.

There are other types of orbits, yet these are the most important.

1.2 Image Processing Pipeline for Space Debris

The Department of Astronomy and Astrophysics (DAA), Faculty of Mathematics, Physics and Informatics of Comenius University in Bratislava, Slovakia developed system for processing space debris and other objects from observations made by 70 cm Newtonian telescope (AGO70, figure 1.3).

Series of consecutive observations are processed by the system to find and catalogue moving objects. The system consists of these steps [4], image processing elements:

1. Image reduction
   Removing additive and multiplicative errors from the image caused by CCD camera. These errors are removed by subtracting a *bias* frame and dividig the frame by *flat field* (taken with closed shutter).

2. Background estimation and subtraction
   Sigma clipping [5] is used to estimate and subtract background noise from an image.

3. Objects search and centroiding (segmentation)
   Objects are detected in an image and their positions with other attributes are saved in text file.

4. Astrometric reduction
   A position of an object from previous step is translated into the equatorial coordinates (section 1.2.1).

5. Masking

Masking is used to remove duplicate objects which appear stationary throughout series like stars.

6. Tracklet building

   Tracklets moving objects are created in this process. This image processing element is explained in detail in the section 2.1 and it is the main focus of this thesis.

7. Object identification

   Identified objects are catalogue.

System input are series of FITS images - Flexible Image Transport System.

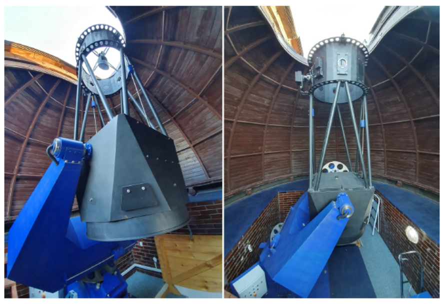


Figure 1.3: AGO70 Newtonian telescope at Astronomical and Geophysical Observatory in Modra, Slovakia. Image from [1].

1.2.1 Equatorial coordinate system

Position of an object after *Astrometric reduction* is given in equatorial coordinates. Equatorial coordinate system represents positions of objects on celestial sphere [6].

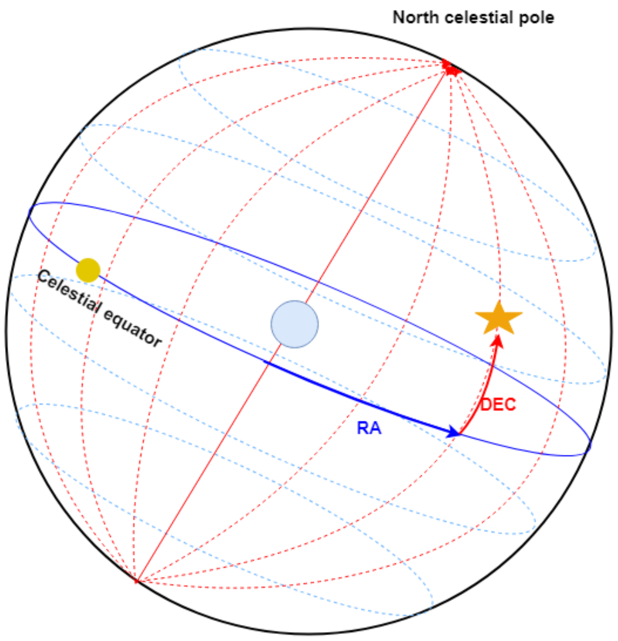


Figure 1.4: Equatorial coordinates of a star on celestial sphere.

Position of an object consists of two coordinates:

- **Declination** - measures the angular distance from celestial equator. Declination has values from -90 up to $+90$. -90 value represents the South pole and $+90$ the North pole.
- **Right ascension** - measures angular distance from Greenwich meridian along celestial equator. Right ascension is usually measured in sidereal hours from $0:0:0$ to $24:0:0$. Right ascension can be transformed to degrees easier manipulation.

Although equatorial coordinates do not represent a position on 2-D plane, our field of observation is very small, and we can assume that equatorial coordinates behave as Euclidean coordinates. Figure 1.4 shows equatorial coordinates on a sphere.

2. Goal of thesis

The goal of the thesis is to improve the **tracklet building** process using neural networks and compare it to the current analytic solution [2]. Analytic solution uses a linear regression to create tracklets. Neural network will be used to find positions of a moving object from the input sequence. This approach should improve system in various ways:

- Without a need to change the design of the algorithm we would be able to detect different types of object trajectories with use of variously trained networks.
- The output of the current system does not include any confidence value. The new algorithm will return confidence value for each tracklet. This could help us detect false positive examples.

2.1 Tracklet building

A tracklet is a data structure containing consecutive observations of an object in time. We can refer to tracklet also as trajectory of the object.

Blue points in the figure 2.1 represent tracklet and red points represent positions which were not assigned to tracklet. We can then approximate these positions by an imaginary line which represents the trajectory

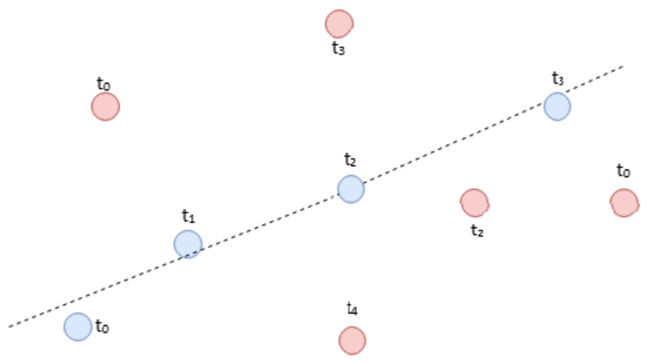


Figure 2.1: A trajectory and tracklet of an object. The object positions are represented by blue points.

of an object in time.

We can approximate an object's trajectory by a line as we are observing only a small part of the sky. Due to a small field of view the object's orbit appears as a line. In this system, simple linear regression (SLR) is used to find a sequence of object positions on a line [4]. Tracklet building is then realized in following steps:

1. Creating Cartesian product of all the objects positions from first and second image.
2. Computing angular velocity and position angle every pair.
3. For each k-tuple we check if there exists an object from next frame which angular velocity and position angle are the closest to the baseline values of the two original objects. This object is added to tracklet.
4. Line parameters for k-tuples are updated with SLR.
5. Continue iteratively from step 3.

After processing objects from all frames, computed tracklets are saved to a text file.

2.2 Deep Learning

In this section I am referencing information from the book *Deep Learning* [7] chapters 1 and 10.

Computers are good in solving problems which can be described formally by mathematical rules. There are problems easy for humans but difficult to be formally described. With lots of training data, deep learning systems can gather knowledge from experience without a need of any prior description. The system is building a hierarchical graph of concepts which allows it to learn complicated concepts from similar inputs. Because this graph of concepts is very deep, we call this approach **Deep Learning**.

2.2.1 Recurrent neural network

Recurrent neural networks (RRNs) are a family of neural networks specialized for processing sequential data. RNNs are made to process sequences of values $x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(n)}$. The sequences can have variable length.

For processing sequences of fixed length we can use simple multilayer perceptron model, however this model is not able to efficiently learn dependencies between parts of input. For example, in language, word order might not change the meaning of sentence.

In last year, there was Covid pandemic.

There was Covid pandemic last year.

Meaning of sentences/sequences is the same but the target words *Covid pandemic* are not in the same place. We want the network to accumulate knowledge from previous input words in processing of the next word.

In RNNs, function of previous output and current input is used for each input. This process stores information from previous time points in a so called **hidden state** $h^{(t)}$. We can better imagine this process by unfolding neural network in time (figure 2.2).

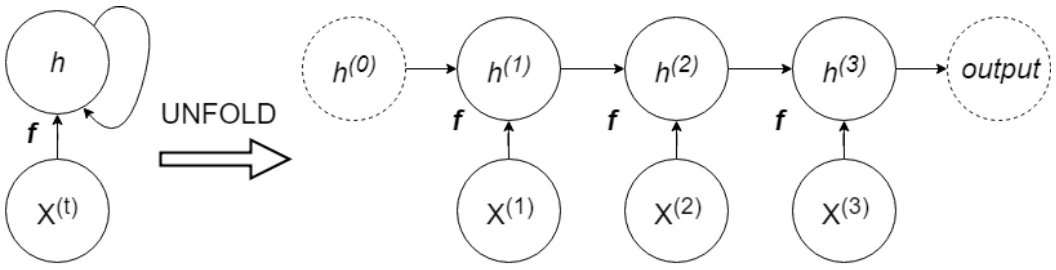


Figure 2.2: Recurrent neural network processing information of length 3.

Each node in the unfolded graph represents network node with input from sequence in time. We can rewrite one step computation as:

$$h^{(t)} = f(h^{(t-1)}, x^{(t-1)}, \Theta)$$

It is proven that any function computable by a Turing machine can be computed by RNN of a finite size.

Leaky recurrent neural network

Problem with big RNNs is vanishing or exploding gradient. Unfolded graph of RNN acts like multi-layer perceptron. If sequences are long, our imaginary network is very deep. It is likely that the gradient will vanish in backward pass (algorithm does not learn) , or explode and focus only on one example. Leaky RNNs have lateral connections in time to avoid this effect (figure 2.3). They accumulate information over long a time.

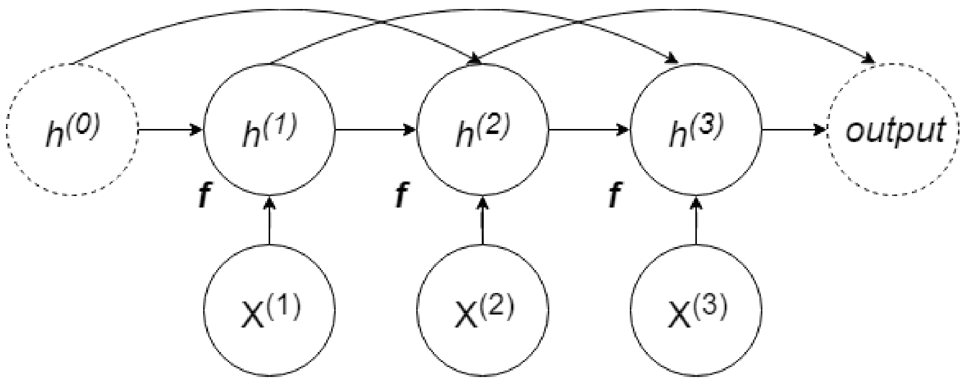


Figure 2.3: Recurrent neural network with lateral connections every second node.

3. Proposed methods

3.1 Data generator

Machine learning models need a fair amount of data examples for training. In the supervised learning, each training example consists of an input and a target value. In our case the input data is series of images captured by the AGO70 telescope and the target value is a tracklet of an object. For training we need processed image sequences int the form of tracklets. However, there is not enough processes sequences, if we want to train a neural network we need to produce artificial training data.

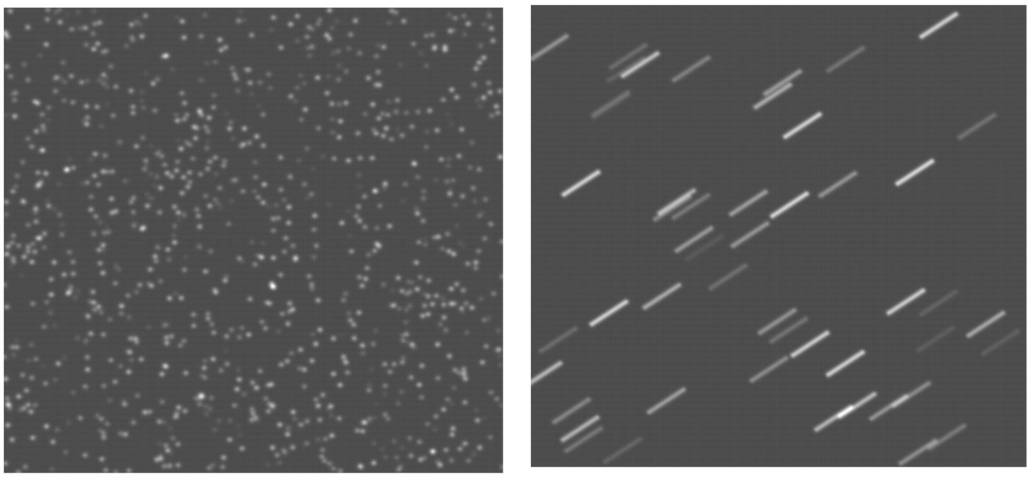
3.1.1 starGen.r

The Department of Astronomy and Astrophysics has provided their own solution to a problem of artificial images generation. **StarGen** script is written in R language and is capable of generating a single FITS image. It has several adjustable parameters:

- **dimX, dimY** - representing size of image
- **starCount** - number of generated stars
- **fwhm** - full width at half maximum (size of star)

- **briMin, briMax** - brightness parameters
- **method** - method used for star generation: gauss, cauchy, line

The script is able to generate FITS images with set number of stars. Stars are represented as a point light source (figure 3.1a) or a streak light source (figure 3.1b), which simulates rapid telescope movement.



(a) Image with 1000 stars and fwhm 10.(b) Image with lines, 10 fwhm, 50 stars.

Figure 3.1: Images generated by starGen.r script.

3.2 SVM method

Support-vector machine (SVM) model is supervised machine learning model used for classification. It projects original input data into higher dimension and performs maximal-margin classification in this higher dimension space (40 Algorithms Every Programmer Should Know [8] chapter SVM algorithm).

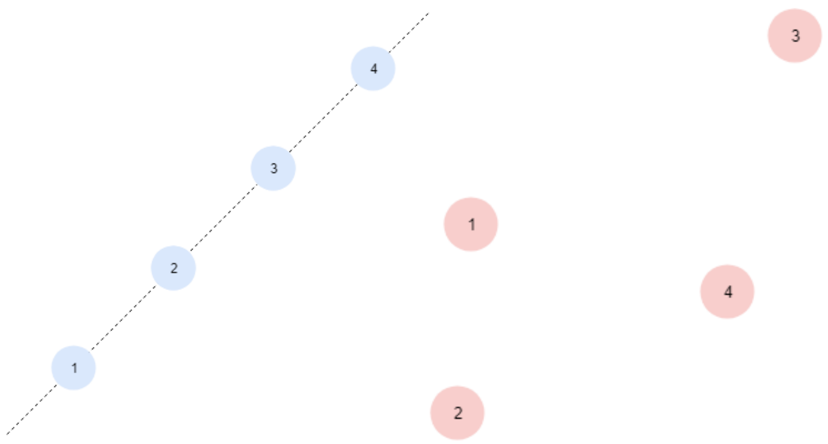
We can use this property and train SVM to classify input tuples of size k into two classes:

1. **On line**

   This class represents tuples with size k and with their points laying on a line with same distance between points in the tuple. Each two adjacent points in a tuple have to be equidistant and lay on the same straight line as in figure 3.2b.

2. **Not on line**

   This class represents tuples with size k, which are not *on line* as in figure 3.2a.



(a) On line class example (b) Not on line class example

Figure 3.2: Example of members of two classes. Blue points represent tuple in On Line class and red points represents tuple in Not On Line class.

3.3 RNN method

RNNs are commonly used for time series forecast and in natural language processing. Sentence word order has a huge effect on a meaning of the sentence in many languages. In RNNs, information about previous inputs is accumulated in the hidden state. This information is used to predict next point in time or next word in sentence (Deep Learning, [7] chapter 10).

We can use LSTM (Long short-term memory) cells to predict current position of a moving object from previous positions. We use the predicted positions to build tracklet of a moving object. According to article ‚*MX-LSTM: mixing tracklets and vislets to jointly forecast trajectories and head poses* [9], LSTM is good choice for this problem.

3.3.1 Long short-term memory - LSTM

Leaky RNNs accumulate information over long period of time. However, in some cases we want to forget part of information we learned. For example, if sequence consists of subsequences. Information from previous subsequence accumulated in the hidden state can lead to wrong prediction about next subsequence. This problem was solved by introducing gated units like **LSTM** (figure 3.3). A gated unit uses gates which control amount of information passing through.

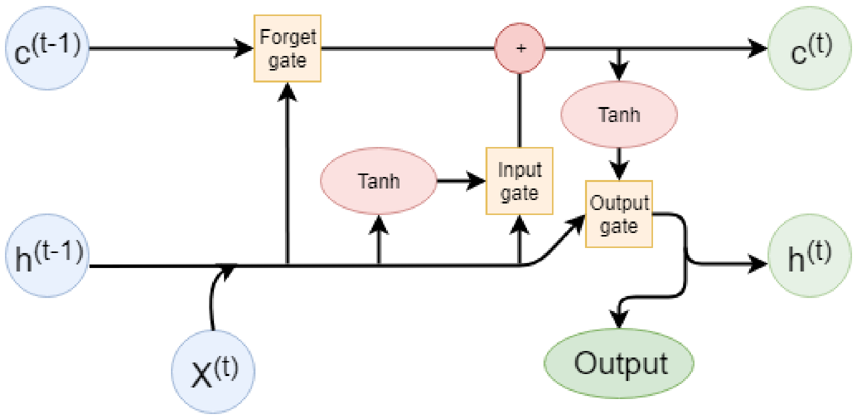


Figure 3.3: Illustration of LSTM unit. Arrows represent information flow. Blue cells represent inputs, green cells represent outputs. Red cells stay for simple operation on data. Yellow cells represent gates.

LSTM has two inner states.

- $c^{(t)}$ represents the **cell state** - the cell state is a path for information to run down through whole network like lateral connections in leaky units
- $h^{(t)}$ represents the **hidden state** - the hidden state is the previous output of LSTM.

LSTMs also have three gates: input, output and forget gate. Each gate is a pointwise multiplication of first input and output of nonlinear function from second input (figure 3.4). A gate has two weight matrices U for input vector and W for hidden state vector.

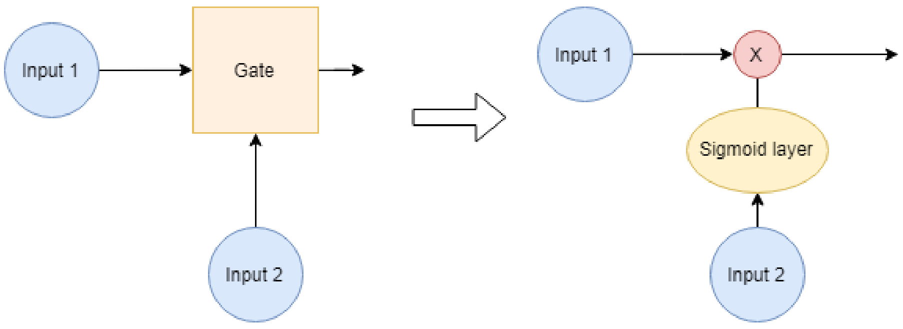


Figure 3.4: Gate.

An input $x^{(t)}$ is at first concatenated with hidden state $h^{(t-1)}$ and goes with $c^{(t-1)}$ through the forget gate. Let $f^{(t)}$ be the output from sigmoid layer from the gate:

$$f^{(t)} = \sigma \left(bias^f + \sum_j U^f_{i,j} x^{(t)}_j + \sum_j W^f_{i,j} h^{(t-1)}_j \right) \tag{3.1}$$

The next cell state is then computed:

$$I^{(t)} = \sigma\left(bias^I + \sum_j U^I_{i,j} x^{(t)}_j + \sum_j W^I_{i,j} h^{(t-1)}_j\right) \tag{3.2}$$

$$l^{(t)} = tanh\left(bias^l + \sum_j U^l_{i,j} x^{(t)}_j + \sum_j W^l_{i,j} h^{(t-1)}_j\right) \tag{3.3}$$

$$c^{(t)} = f^{(t)} c^{(t-1)} + I^{(t)} l^{(t)} \tag{3.4}$$

where $I^{(t)}l^{(t)}$ is the output of the input gate and $l^{(t)}$ is the LSTM internal unit.

Next hidden state is computed

$$O^{(t)} = \sigma\left(bias^O + \sum_j U^O_{i,j} x^{(t)}_j + \sum_j W^O_{i,j} h^{(t-1)}_j\right) \tag{3.5}$$

$$h^{(t)} = tanh(c^{(t)})O^{(t)} \tag{3.6}$$

Let n be the size of the input vector and m size of the hidden state vector. At the end, the LSTM unit consists of four pairs of matrices - W with shape $m \times m$ and U with shape $m \times n$. The final number of trainable parameters is then:

$$N = 4 \cdot (n \times m) + 4 \cdot m^2 \tag{3.7}$$

LSTM networks have been shown to learn long-term dependencies more easily than the simpler recurrent neural networks. LSTMs are good for forecasting time series.

One disadvantage of LSTMs is the training time. Using time folding with this many parameters causes long training time of LSTMs, even with relatively small input and small hidden dimensions.

3.4 Encoder-decoder method

Another possible solution is to use the **encoder-decoder** architecture for trajectory prediction according to the article *Sequence-to-Sequence Prediction of Vehicle Trajectory via LSTM Encoder-Decoder Architecture* [10].

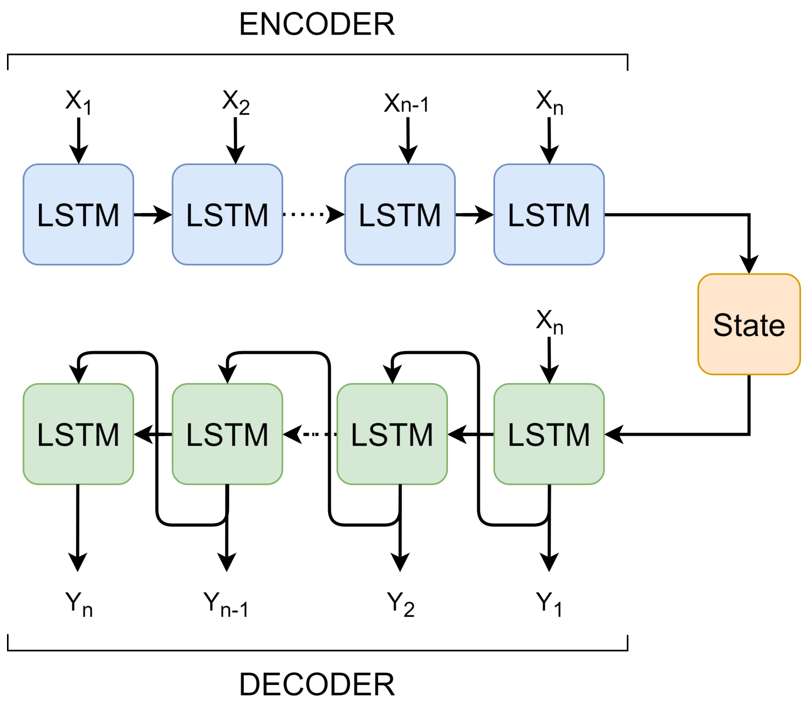


Figure 3.5: The encoder-decoder architecture of a neural network. Graph shows how an output sequence is computed from an input sequence.

This neural network consist of two main components:

1. **Encoder**

   The encoder is a recurrent neural network. It encodes an input sequence into the hidden state. Information about whole sequence is accumulated in the hidden state. In figure 3.5 LSTM units are used as recurrent units.

2. **Decoder**

   The decoder is a recurrent neural network. It is made out of several recurrent units and often a linear output layer. The decoder decodes last output and the hidden state to predict the next output. In figure 3.5, $y_i, i = 1..n$ represent outputs and $x_i, i = 1..n$ represent inputs. LSTM units are used as recurrent units in the figure 3.5.

This architecture allows to model more complex object trajectories. Simple RNN method is good for linear trajectories. The encoder-decoder could give us better precision for more complex trajectories.

This method was not implemented due to sufficient performance of the RNN method.

4. Software design

The tracklet building system is an algorithm described by activity diagram 4.1. An **input sequence** contains a list of positions for each corresponding image. **Masking** processes the sequence to remove stationary objects like stars from the sequence. From the first two image's lists we **create tuples**. Tracklets are created from these tuples. The **model predicts** the next position for each tracklet (position in the next image). The **Matching function** finds the closest positions from the next image to predicted positions. If the function is not able to find any position within the threshold, the tracklet would have marked this position as missing. If we have already predicted **the last position** we return remaining tracklets. Otherwise, if at least **k positions in row are marked as missing** in tracklet we discard that tracklet. Remaining tracklets are updated. The **Tracklet update** updates tracklet's confidence. Then the model will predict positions for tracklets again.

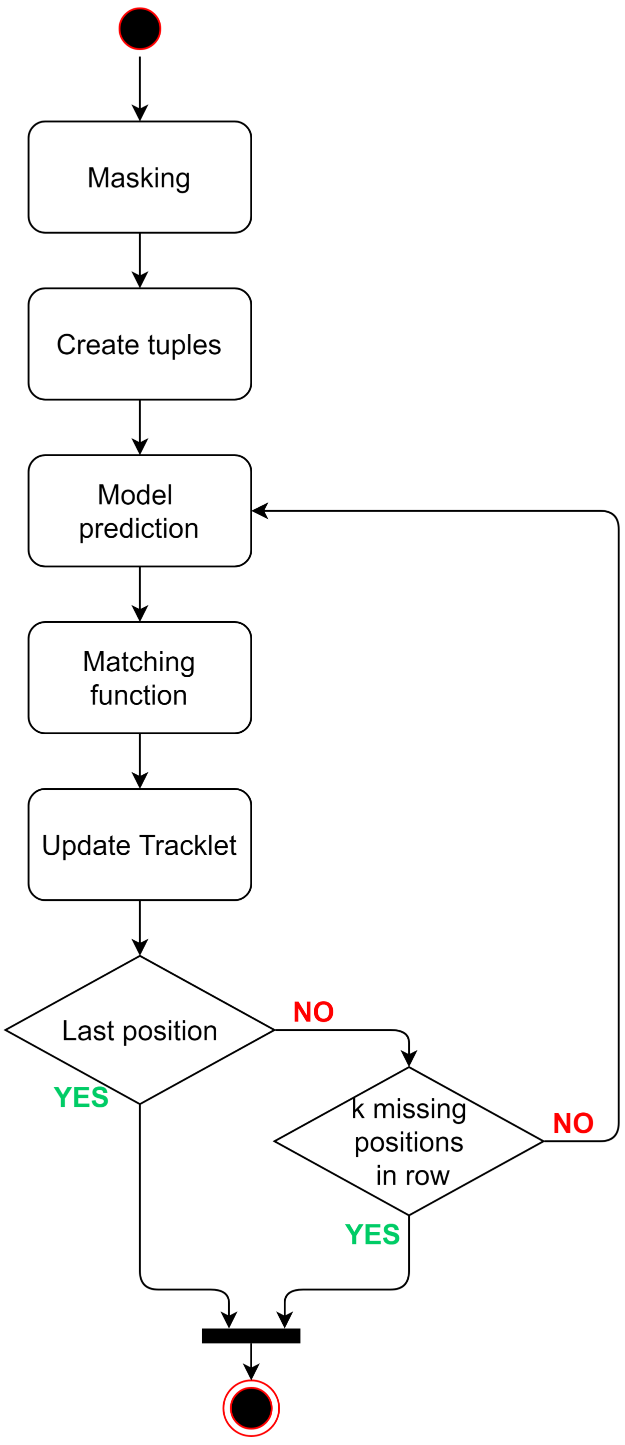
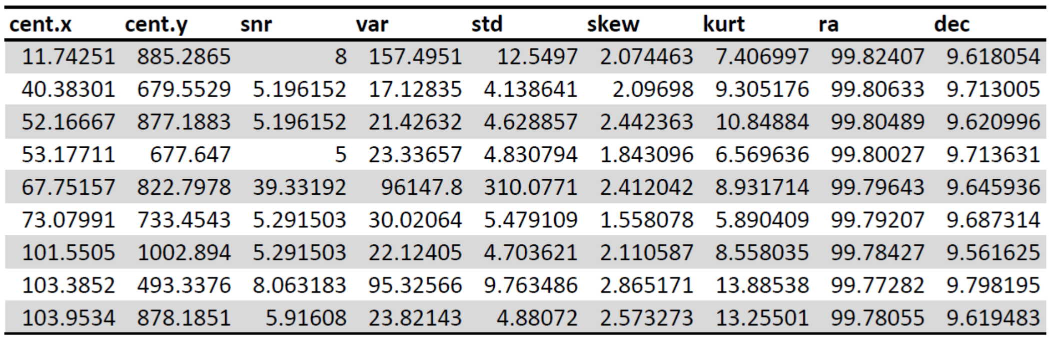


Figure 4.1: The activity diagram of the tracklet building system. A system input is sequence and tracklet is the output.

4.1 Input and output data

4.1.1 Input

Input data is a sequence consisting of lists. Each list contains positions of objects in an image. For each image in a sequence corresponds to one list of positions. Lists are ordered by the image's capture time.



cent.x	cent.y	snr	var	std	skew	kurt	ra	dec
11.74251	885.2865	8	157.4951	12.5497	2.074463	7.406997	99.82407	9.618054
40.38301	679.5529	5.196152	17.12835	4.138641	2.09698	9.305176	99.80633	9.713005
52.16667	877.1883	5.196152	21.42632	4.628857	2.442363	10.84884	99.80489	9.620996
53.17711	677.647	5	23.33657	4.830794	1.843096	6.569636	99.80027	9.713631
67.75157	822.7978	39.33192	96147.8	310.0771	2.412042	8.931714	99.79643	9.645936
73.07991	733.4543	5.291503	30.02064	5.479109	1.558078	5.890409	99.79207	9.687314
101.5505	1002.894	5.291503	22.12405	4.703621	2.110587	8.558035	99.78427	9.561625
103.3852	493.3376	8.063183	95.32566	9.763486	2.865171	13.88538	99.77282	9.798195
103.9534	878.1851	5.91608	23.82143	4.88072	2.573273	13.25501	99.78055	9.619483

Figure 4.2: Output .tsv of astrometric reduction from the current system.

The figure 4.2 is an example of a TSV (tab-separated values) input file. One TSV file contains frame objects from one observation in time. Not all fields in the TSV are displayed. Each row represent an object in the image. We only need information about positions of objects. Lists in input sequence contain only the last two fields from TSV files: RA (right ascension), DEC (declination). These fields represent positions of objects in equatorial coordinate system.

In the figure 4.3, there is an example of an input sequence. Each list contains only few positions for demonstration purposes.

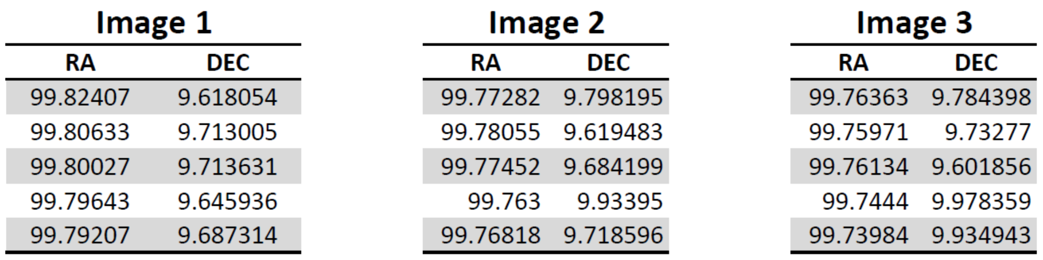
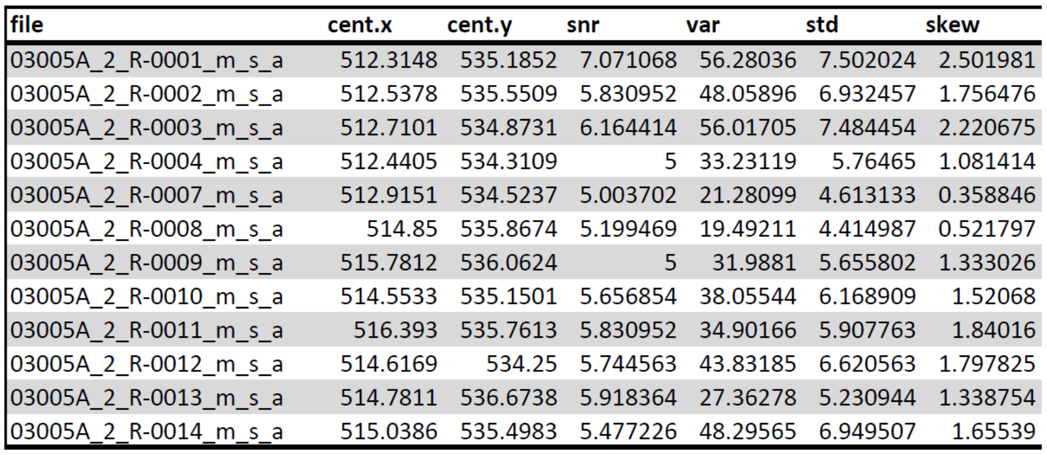


Image 1		Image 2		Image 3	
RA	DEC	RA	DEC	RA	DEC
99.82407	9.618054	99.77282	9.798195	99.76363	9.784398
99.80633	9.713005	99.78055	9.619483	99.75971	9.73277
99.80027	9.713631	99.77452	9.684199	99.76134	9.601856
99.79643	9.645936	99.763	9.93395	99.7444	9.978359
99.79207	9.687314	99.76818	9.718596	99.73984	9.934943

Figure 4.3: An example of input sequence. Image i represents list of positions of objects in time i from one observation.

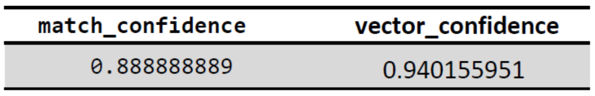
4.1.2 Output

The output of the system are two TSV files representing one tracklet. The first file with name "*<sequence_name>_TB_number.tsv*" contains information about a found object positions. The second file with name "*<sequence_name>_TB_%2d_confidence.tsv*" contains confidence numbers for this tracklet, which are described in section 4.6. If multiple moving objects were found in one sequence the system produce a pair of files for each one.



file	cent.x	cent.y	snr	var	std	skew
03005A_2_R-0001_m_s_a	512.3148	535.1852	7.071068	56.28036	7.502024	2.501981
03005A_2_R-0002_m_s_a	512.5378	535.5509	5.830952	48.05896	6.932457	1.756476
03005A_2_R-0003_m_s_a	512.7101	534.8731	6.164414	56.01705	7.484454	2.220675
03005A_2_R-0004_m_s_a	512.4405	534.3109	5	33.23119	5.76465	1.081414
03005A_2_R-0007_m_s_a	512.9151	534.5237	5.003702	21.28099	4.613133	0.358846
03005A_2_R-0008_m_s_a	514.85	535.8674	5.199469	19.49211	4.414987	0.521797
03005A_2_R-0009_m_s_a	515.7812	536.0624	5	31.9881	5.655802	1.333026
03005A_2_R-0010_m_s_a	514.5533	535.1501	5.656854	38.05544	6.168909	1.52068
03005A_2_R-0011_m_s_a	516.393	535.7613	5.830952	34.90166	5.907763	1.84016
03005A_2_R-0012_m_s_a	514.6169	534.25	5.744563	43.83185	6.620563	1.797825
03005A_2_R-0013_m_s_a	514.7811	536.6738	5.918364	27.36278	5.230944	1.338754
03005A_2_R-0014_m_s_a	515.0386	535.4983	5.477226	48.29565	6.949507	1.65539

Figure 4.4: File 03005A_2_R_TB_00.tsv. It contains information about positions from tracklet found in 03005A_2_R_27-11-20_10..30..35 series.



match_confidence	vector_confidence
0.888888889	0.940155951

Figure 4.5: File 03005A_2_R_TB_00_confidence.tsv. It contains tracklet's confidence numbers.

The figures 4.4 and 4.5 represent output files pair for sequence **03005A_2_R_27-11-20_10..30..35**.

4.2 Masking

The current system can produce multiple positions for one object in the picture during segmentation stage. The images contain stars, which have approximately at the same positions throughout the whole sequence. We reduce the number of positions in the sequence by removing stationary stars positions and multiple object positions. With smaller number of positions, it is easier and faster to find positions of an moving object in a sequence.

4.2.1 Algorithm

The Department of Astronomy and Astrophysics has provided algorithm used in the current system described in article *Selected Modules from the Slovak Image Processing Pipeline for Space Debris and Near Earth Objects Observations and Research* [4]. Masking function is described by pseudocode 4.1. We create list of positions from whole input sequence. For each position we remember from which image it comes from and create *retain flag*. The retain flag is initially set to False. If it is set to True, positions will be used in output sequence. We set *current* position to be the first position from ordered list of positions. We remove all positions from input sequence which are no farther than threshold from the current position. The first position with distance greater than threshold is the new current position.

Procedure 4.1 Masking function

```
Input: sequence, threshold
Output: new_sequence
 1: positions ← positions from whole sequence
 2: retain_flag ← boolean flag for each position in positions
 3: ordered_positions ← sorted positions by $x$ and $y$ value
 4: current ← first position - positions[1]
 5: mark current in retain_flag
 6: for next in ordered_positions do
 7:    dist ← Euclidean position between current and next
 8:    if  dist > threshold  then
 9:       current ← next
10:       mark next in retain_flag
11:    end if
12: end for
13: new_sequence ← retain marked positions from original sequence
```

This approach will remove stationary stars and multiple position objects if threshold value is properly set.

4.3 Create tuples

We create tuples with size 2 from positions from first two lists in a sequence. These pairs are created as Cartesian product between positions from the first and the second image. For this purpose, we can use Python module *itertools* with function *product*. As in the figure 4.6, we can create Cartesian product of two lists by calling *product(list1, list2)*.

```
>>> a = ['a', 'b', 'c']
>>> b = ['d', 'e', 'f']
>>> import itertools
>>> list(itertools.product(a, b))
[('a', 'd'), ('a', 'e'), ('a', 'f'), ('b', 'd'), ('b', 'e'),
('b', 'f'), ('c', 'd'), ('c', 'e'), ('c', 'f')]
```

Figure 4.6: Use of itertools.product on example.

4.4 Model prediction

Model is a trained neural network. For each tuple of positions and tracklet, model outputs predicted position in the next image. An input to model has to be normalized. Also, the model output is normalized. To get real positions in an image we need to denormalize this positions. We normalize data by subtracting minimum for each coordinate and dividing by variance for each coordinate. Variance is computed as difference between minimum and maximum for each coordinate. Denormalization is inverse operation.

$$normalize(x) = \frac{x - MIN(data)}{MAX(data) - MIN(data)} \tag{4.1}$$

$$denormalize(x) = x \cdot (MAX(data) - MIN(data)) + MIN(data) \tag{4.2}$$

4.4.1 Network topology

To function properly, neural network has to be designed well. For a trajectory forecast we chose a network with LSTM cells. It accumulates information about the movement of an object from its previous positions to predict current position in the image.

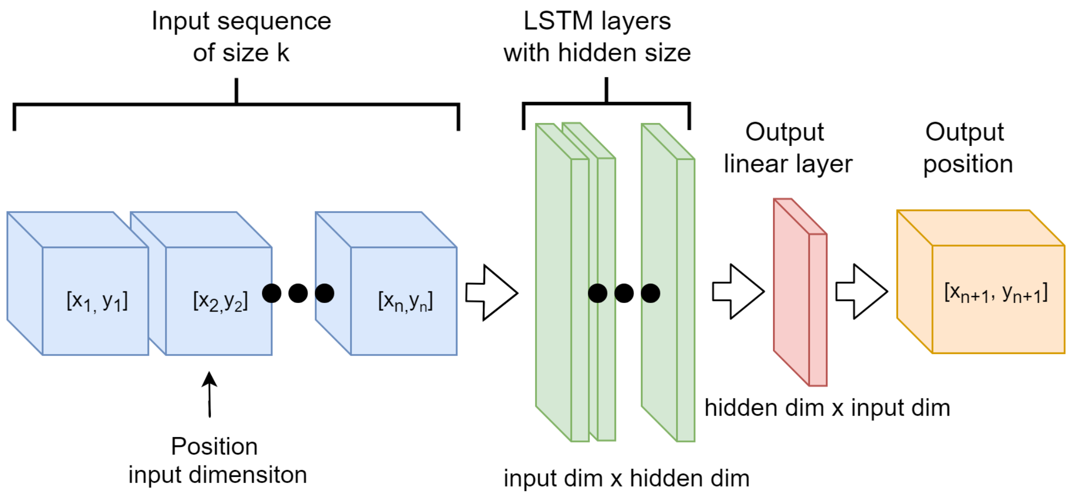


Figure 4.7: Topology of the neural network. Input sequence is processed by LSTM layers. Output is produced by linear layer to transform LSTM output to input dimension space.

The neural network topology is illustrated in figure 4.7, its implementation is in figure 4.8. Positions from an input sequence are fed into LSTM layers. LSTM projects points from **input dimension** into **hidden dimension** space. Before entering the output linear layer, a nonlinear activation function is applied (tanh, ReLU). Output layer projects data from hidden dimension to input dimension space. The result is a vector of the same dimension as the input.

```
class AlbertNet(nn.Module):
  def __init__(self, hidden_size, input_size=2, num_layers=1):
    super(AlbertNet, self).__init__()
    self.LSTM = nn.LSTM(input_size,hidden_size, num_layers)
    self.final = nn.Linear(hidden_size, 2)

  def forward(self, seq):
    output, (hidden,_) = self.LSTM(seq)
    output = torch.tanh(output[-1,:,:])
    pred = self.final(output)
    return pred
```

Figure 4.8: Declaration of AlbertNet neural network in Pytorch.

4.5 Matching function

Matching function assigns position from the next list to each predicted position.

4.5.1 Algorithm

As we can see in the pseudocode 4.2, for each predicted position the algorithm finds position from the next image closest to the predicted within the threshold.

Procedure 4.2 Matching function

```
Input: predictions, image_positions, threshold
Output: matched_positions
 1: matched_positions ← empty list
 2: for prediction in predictions do
 3:     candidates ← empty list
 4:     for candidate in image_positions do
 5:         dist ← Euclidean distance between candidate and prediction
 6:         if dist < threshold then
 7:             add (dist,candidate) to candidates
 8:         end if
 9:     end for
10:     if candidates is empty then
11:         position ← $[-1, -1]$ invalid position
12:     else
13:         position ← position with minimum distance from candidates
14:     end if
15:     add position to matched_positions
16: end for
```

The threshold value determines behaviour of whole the system. Small values can cause tracklet to not be found. On the other hand, if the value is too large, system could produce tracklets with small vector confidence value.

4.6 Tracklet update

During processing, potential tracklets are held in *Tracklet* class. Tracklet contains predicted and matched positions in order. It also contains confidence values for itself. Tracklet update recomputes match confidence number and vector confidence number with new information from last prediction.

Match confidence number from 4.5 represents proportion of found positions to total number of positions. **Vector confidence** number from 4.5 represents average of difference between the norm of true vector $\vec{\boldsymbol{t}}$ and the norm of difference between the true and prediction vector $\vec{\boldsymbol{p}}$ all divided by norm of true vector.

$$\text{vector\_confidence} = \frac{|\|\vec{\boldsymbol{t}}\| - \|\vec{\boldsymbol{t}} - \vec{\boldsymbol{p}}\||}{\|\vec{\boldsymbol{t}}\|} \tag{4.3}$$

4.7 Parameters

4.7.1 Masking threshold

If the distance between two points during masking process is equal or less than the masking threshold, these points are considered to be one object. Number of thrown away positions during masking depends on the threshold value. The bigger the value is the more positions are thrown away during masking.

In our program, the threshold stays for distance in percent of difference between maximal and minimal position. If $threshold = 0.01$ and the $difference = (100, 100)$, then the threshold value is $(0.1, 0.1)$. Value of parameter does not depend on position units.

4.7.2 Matching threshold

The matching threshold tells us the maximum distance between predicted position and position in an image. This value can influence total amount and accuracy of found tracklets. The bigger the threshold is, the more tracklets will be found, but many with low confidence.

Like the masking threshold, the matching threshold is in percent of difference between maximal and minimal position.

4.7.3 Value K

The K value is the maximum number of missed positions in row. Final tracklet could have missing positions from at most K consecutive images. Many times, objects in an image are not recognized by the segmentation algorithm and their positions are not in the input sequence for our system. Therefore, the value of K allow system to find tracklets with missing positions. If the value is too big, system could find random tracklets thanks to many missing position.

4.7.4 Overlap

The system can only process sequences with maximum length of 8. Yet, test data sequences can have varying length depending on observation strategy. We need to find smaller tracklets in parts of sequence and then connect them into one. Overlap value tells us overlap between processed subsequences of input sequence.

5. Research

5.1 Data generator - StarGen.py

The script *starGen.r* generates images which cannot be directly used as training examples for our machine learning models. Therefore, whole script was re-written in Python and modified. The way how the trajectory of moving objects in training data is calculated is crucial for model behaviour. A trained model will be able to recognize only objects with this type of trajectory.

- **Series of images**

  For training data we need series of images. New generator generates image series with length 8 (figure 5.1).

- **Objects**

  The original script does not simulate a space debris object, or anything else beyond the star field. We have modified the script to produce variable number of moving objects. Their trajectories consist of $[x, y]$ positions in each image of the series. We can see movement of object in figure 5.2.

- **TSV output file**

  Positions of both stars and objects in each frame are stored into *TSV* (figure 5.3) file with same structure as after the **Segmentation** step in the existing system (see section 1.2). This ensures that artificial data hold the same structure as real data. Files contain one extra boolean field for marking moving object's positions.

- **Point objects**

  For training purposes and scope of this thesis, script is generating only point light sources.

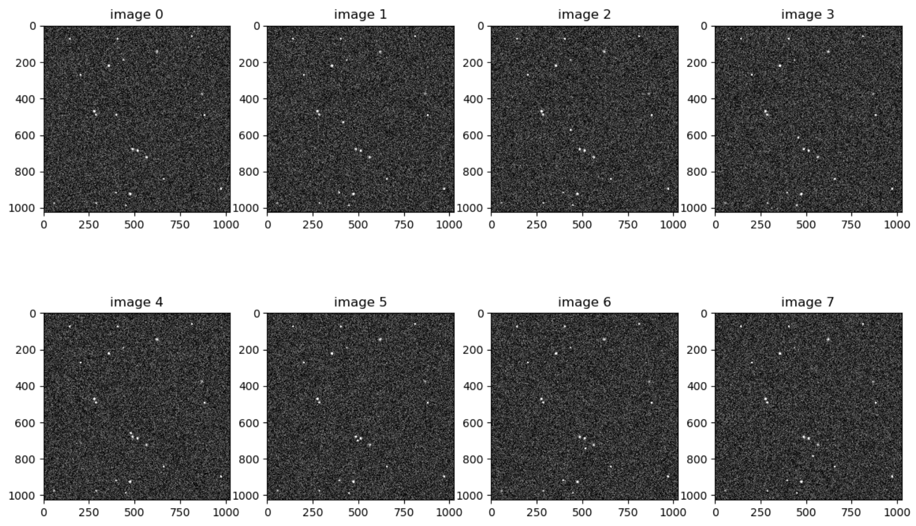


Figure 5.1: Series of 8 images created by generator. Series contains one moving object and random number of stars from the uniform distribution between 1 and 50.

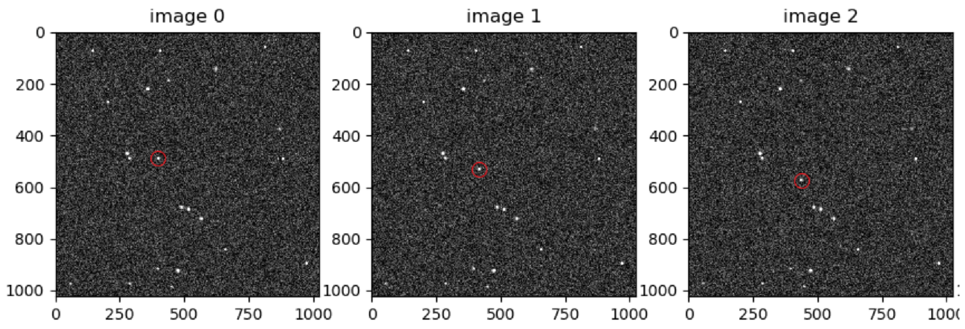
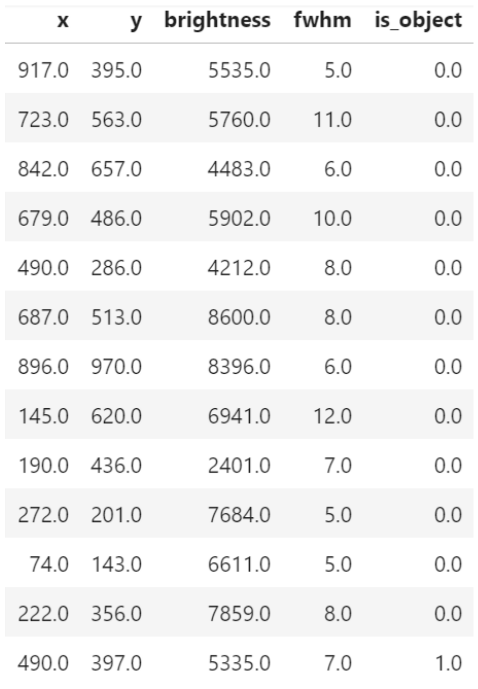


Figure 5.2: The first three images from series. Moving object is in the red circle and it is moving downwards.



x	y	brightness	fwhm	is_object
917.0	395.0	5535.0	5.0	0.0
723.0	563.0	5760.0	11.0	0.0
842.0	657.0	4483.0	6.0	0.0
679.0	486.0	5902.0	10.0	0.0
490.0	286.0	4212.0	8.0	0.0
687.0	513.0	8600.0	8.0	0.0
896.0	970.0	8396.0	6.0	0.0
145.0	620.0	6941.0	12.0	0.0
190.0	436.0	2401.0	7.0	0.0
272.0	201.0	7684.0	5.0	0.0
74.0	143.0	6611.0	5.0	0.0
222.0	356.0	7859.0	8.0	0.0
490.0	397.0	5335.0	7.0	1.0

Figure 5.3: The final TSV file representing positions of stars and objects from first image of series in figure 5.1.

Parameters for the script are stored in *config.yml* file. We can set: number of series, number of objects and stars in frame, stars and object paramaters. Script produces configurable number of series consisting of eight FITS images and eight TSV files.

5.2 SVM method

5.2.1 Algorithm

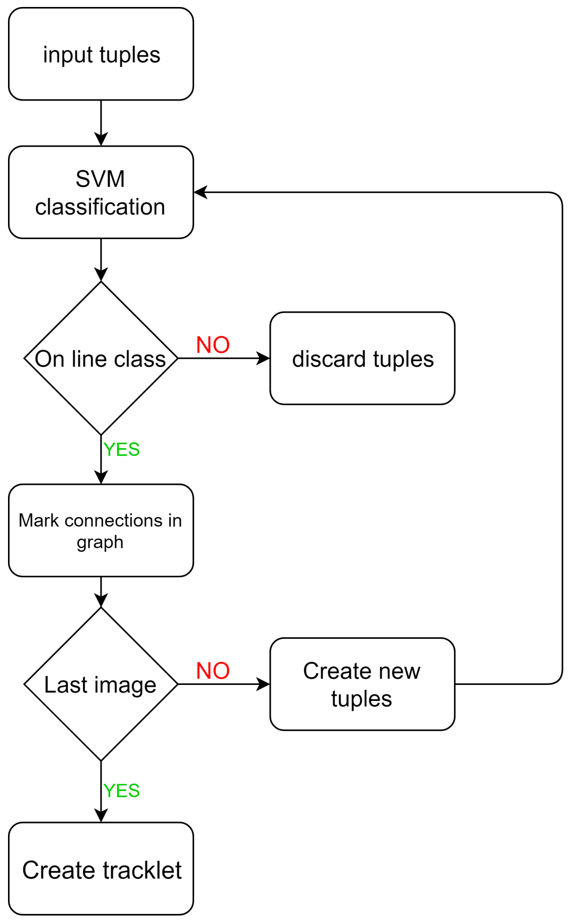


Figure 5.4: Activity diagram of algorithm which is using SVM for classification.

Algorithm is described by the activity diagram (figure 5.4). **Input tuple** consist of k positions $[x, y]$. The first position of an object in the tuple is position from i-th image's TSV file, second from $(i + 1)$-th file and so on. Last position in the tuple is from $(i + k)$-th image's TSV file. Initially, tuples are created by applying Cartesian product between positions of objects from the first k lists. This process will create n^k tuples if n is the number of positions in one list. Tuples are classified with the **SVM classifier** into one of two classes: On Line class and Not on Line class. **Not on Line** class tuples are discarded and further proceed only **On Line** class tuples. If the result of the classifier is *On Line*, connections between positions in the tuple are marked in graph. If last element in a tuple is not a position from the last list, **new tuples are created**. First element is removed from the tuple, and at the end a position from the next list is appended. New tuple will be created for every position from last list.

After processing all input tuples, we find path from first list positions to last list positions. If such path exists, we consider it as **tracklet**. Positions in the path are positions of moving object.

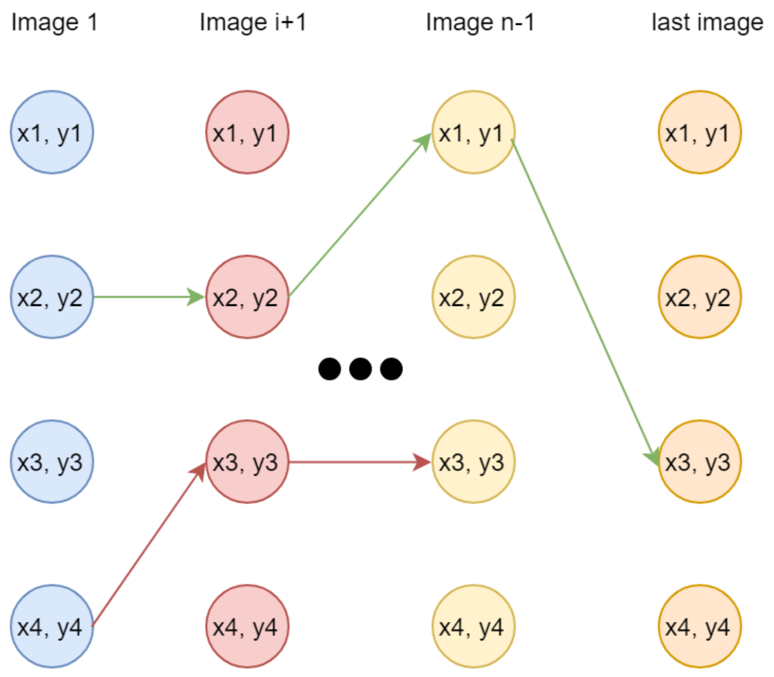


Figure 5.5: Possible connections in final graph after processing whole image sequence. Arrows represents path between positions.

In figure 5.5 there are two possible paths. The green path leads from first list to last list and can be therefore declared as a tracklet. Nodes in the graph represent positions of moving object in tracklet. Red path is an example of an "incomplete" tracklet.

5.2.2 Disadvantages

Finding moving objects using SVM method turns out to be unreliable. While this method is simple, it comes with a cost.

Parameter m and creation of input tuples

Parameter m represents the size of input tuples to SVM classification. Smaller value than $m = 3$ cannot be used, because small values would cause all tuples to fulfill the condition for On Line class. As value of m is higher, the classification is more precise. Yet, high value of m leads to high number of input tuples. Let n be the number of positions in one image, then n^m is the number of input tuples. Value of n can be in hundreds, and computation becomes very slow for even slightly higher values of m such as 4 or 5.

Big number of Not On Line class in real data

SVM classifier trained on 50 thousand input tuples with size $m = 3$ had above 96% accuracy. Only 3% of validation data were classified as false negative (figure 5.1). In the real world data, if only one moving object was in sequence, every input tuple except the one containing positions of moving the object would belong to Not On Line class. This disproportion between classes leads to a big number of false negatives. As a result, we classify many tuples as On Line class, but only one truly belongs there.

		True class	
		On Line	**Not On Line**
Predicted class	**On Line**	4996	4
	Not On Line	363	4637

Table 5.1: Confusion Matrix. 10 000 examples with size m=3 were classified. Only 367 of them were classified incorrectly.

If our algorithm classifies more than one tuple from the last generation (tuples with last element of the last list) as On Line class, we can certainly say that in our graph, there exists at least that many paths from first image to last image. Real number of paths can be much higher.

In testing scenario with $m = 3$, there were 180 tuples from last generation classified as On Line class. That means, we have created at least 179 false tracklets. With $k = 4$ results were similar. Algorithm identified at least 143 false tracklets, and classification had 98% accuracy (figure 5.2).

		True class	
		On Line	**Not On Line**
Predicted class	**On Line**	4997	3
	Not On Line	152	4848

Table 5.2: Confusion Matrix. 10 000 examples with size k=4 were classified. Higher value of k leads to better precision and lower number of misclassified examples.

With this many false negative examples we are not able to detect a true, moving object. Due to these problems SVM method has been rejected.

5.3 Model selection

Network with described topology has two adjustable parameters: **hidden dimension** and **number of LSTM layers**. In order to choose the best parameters, we trained multiple models for 20 epochs (learning iteration) with 500 batches each containing 32 sequences with length 2. Moreover, the specification of input and output form of the network is needed.

5.3.1 Hidden dimension

LSTM projects inputs into hidden dimension space. In order to accumulate enough information in the hidden state, it must be in higher dimension space than input. Higher values than 50 lead to higher training error as shown in the table 5.3. For this reason, we choose the highest value of hidden dimension without significantly increasing the training error after 20 epochs. We set hidden dimension to **hid_dim = 50**.

hidden dim	**training error [px]**
2	72
5	20
15	6
50	6
100	9

Table 5.3: The table shows, how training error is affected by size of the hidden dimension. Training error is Euclidean distance between predicted and the real position in pixels.

5.3.2 Number of LSTM layers

This value represents number of LSTM cells in the network. Using multiple LSTM cells leads to the increase of training time required. The training error is also converging slower as we can see in the table 5.4. For these reasons we chose **number of layers = 1**.

Number of LSTM layers testing		
number of layers	**training error [px]**	**epoch time [s]**
1	6	1.9
2	11	2.55
3	13	3.13
5	18	4.5

Table 5.4: Change in training error and epoch time for different numbers of LSTM layers in network. Training error is Euclidean distance between predicted and true next position in pixels. Epoch time is average duration for training one epoch.

5.3.3 Network input

The LSTM network works with time series, therefore inputs, must be given as a sequences of points. In this case, point represents $[x, y]$ position of a moving object in the corresponding image. One series consists of multiple ordered positions of moving object. An object's positions cannot be missing in the series.

Length of sequences can vary. The longer the input sequence, the more information can the network learn about object's movement. Moreover, humans can more easily predict next position if they have seen multiple previous positions. Training input sequences have been generated using **Data generator**, as we do not have enough real data sequences. Positions of moving objects are only used as the training data. Positions of stars are not required for training task. In the table 5.5 we can see an example of three sequences. In supervised learning, target value is necessary. We use last object's position in series as the target value. We predict last position from series of previous positions.

	image number				
	1	**2**	**3**	**4**	**5**
Series of positions	$[210, 34]$	$[246, 33]$	$[282, 32]$	$[318, 31]$	$[351, 30]$
	$[767, 527]$	$[723, 510]$	$[679, 493]$	$[635, 476]$	$[591, 459]$
	$[624, 580]$	$[670, 593]$	$[716, 606]$	$[762, 619]$	$[808, 632]$

Table 5.5: Example of three series. Each row consists of 5 moving object's positions $[x, y]$ in first 5 images.

After choosing model's parameters we need to specify the input length. To test all possibilities, we trained models with parameters chosen in previous section.

Sequence with length 2

The input sequence consists of the last two positions of the moving object. Two points contain enough information to predict the next position if the object moves in a straight line.

In the figure 5.6, model is learning quickly. Training loss after 20 epochs is around 3 pixels. However, using this approach, we use only a fraction of known information about the object’s movement.

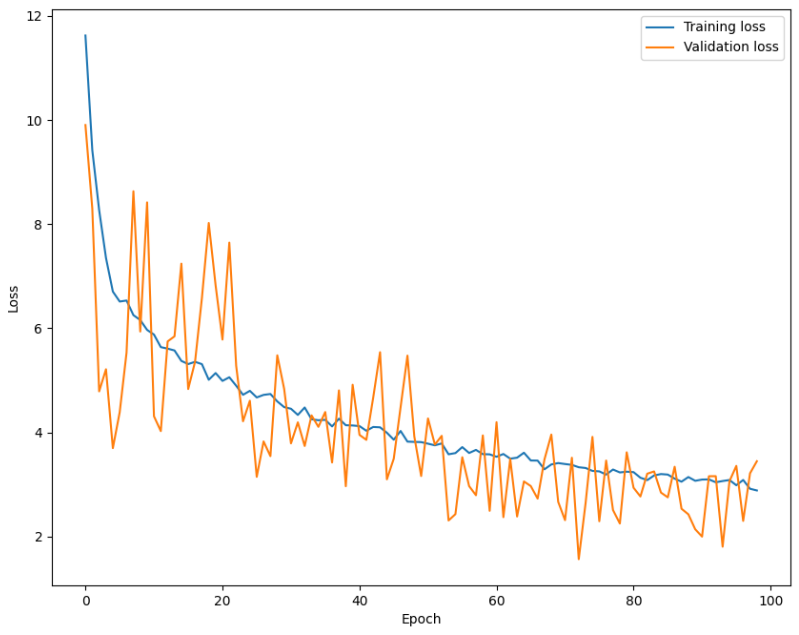


Figure 5.6: Training and Validation loss during training for model using input sequence of length 2.

We trained model for 500 epochs to get the best performance. The behaviour of training and validation loss is visible in the figure 5.7. After 500 epochs training loss is below 2 px.

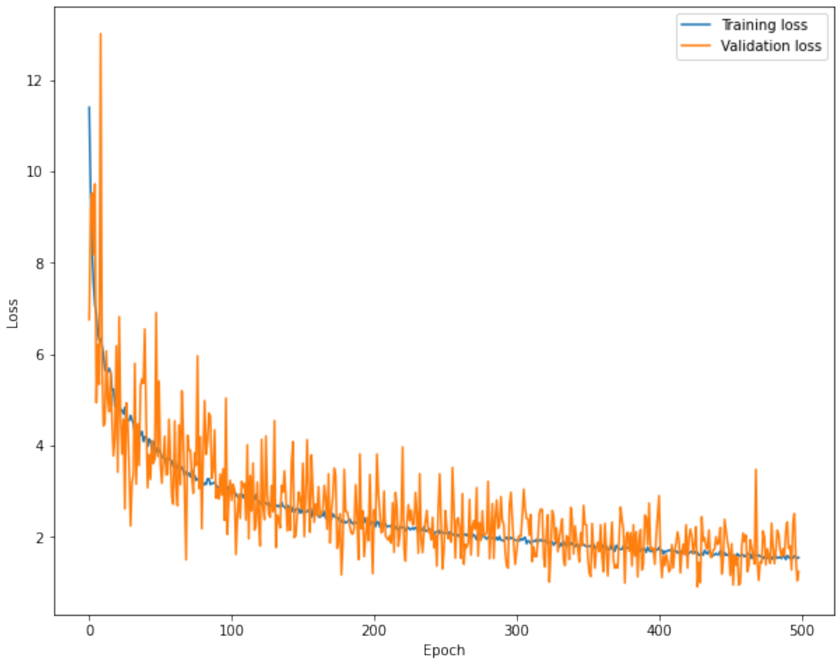
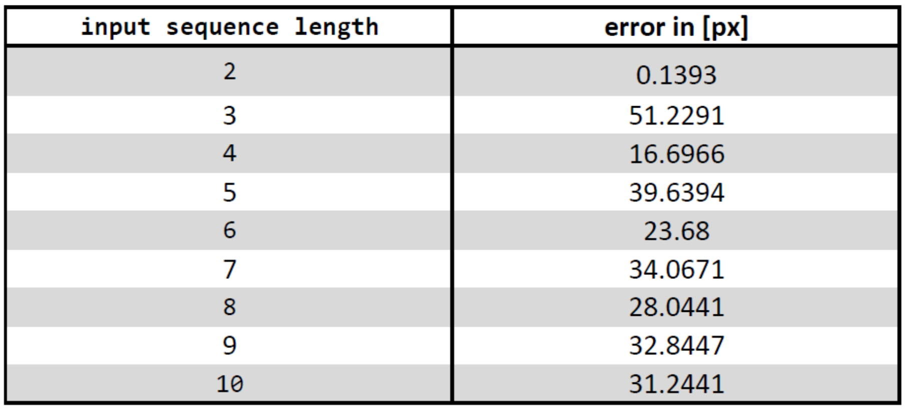


Figure 5.7: Training error for 500 epochs for model with input sequence lenght 2.

A model trained this way cannot be used to process longer sequences. As shown in the table 5.6, testing error is increasing with an input sequence length. Testing error behaviour shows interesting results. Even length values show better results than odd values. This can be caused by training model with even size sequence.

In the figure 5.8 we can see performance of the model using real data. Orange points represent real positions, blue points represent positions predicted by the model. Values represent position number. Real data often do not include all positions of the object. The third real position of object is missing. Due to that, predicted position with number 4 an 5 are distant from real positions. Model performs quite well if no positions are missing. Yet, because of only two previous object's positions have been used, model made relatively big mistake in predicting 9-th position.



input sequence length	error in [px]
2	0.1393
3	51.2291
4	16.6966
5	39.6394
6	23.68
7	34.0671
8	28.0441
9	32.8447
10	31.2441

Table 5.6: Testing error for input sequences with different length. Each length was tested on 100000 examples 5 times to get average error.

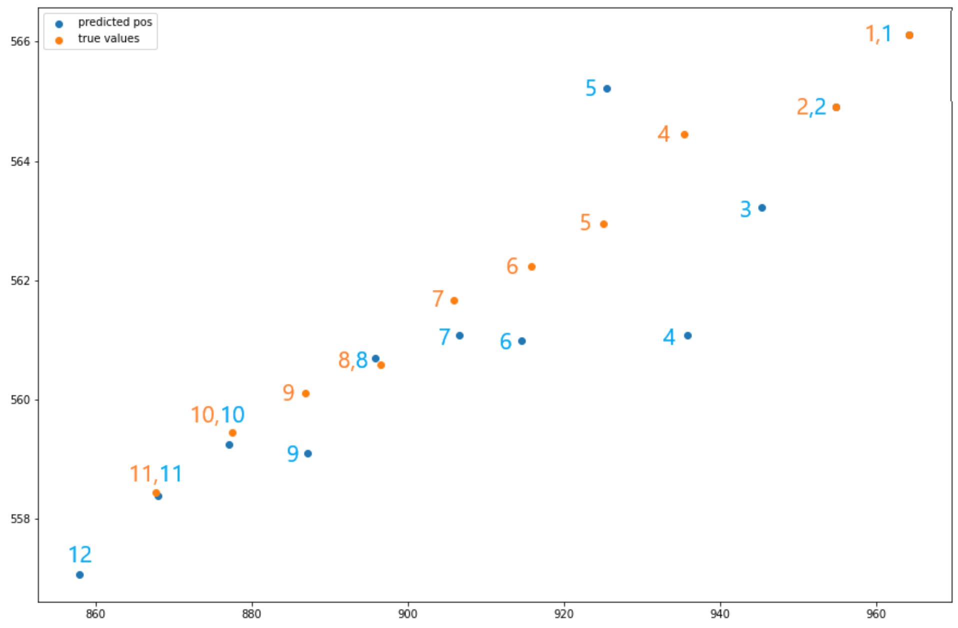


Figure 5.8: Positions of the moving object. Object is moving from top right corner to bottom left. Orange points are positions of moving object. Blue point are predicted positions by model. Position prediction was made by model trained with input sequence length = 2.

Advantages - fast learning, fast prediction

Disadvantages - Uses only partial information about movement of the object. Model is not consistent in precision of predictions.

Variable sequence length

The input sequence consists of all known positions of the object. Using this approach, we use every available information about the object's movement. Thanks to that, model is less sensitive to local deviations in object movement.

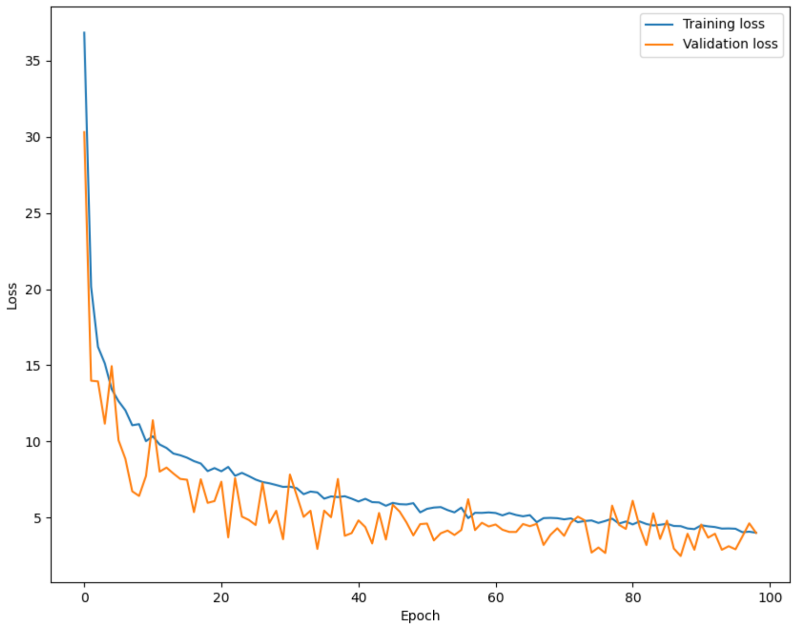


Figure 5.9: Training loss (in blue) and validation loss (in orange) during training for 100 epochs for model trained with variable length input. Training loss reached its minimum at approximately 4 pixel loss.

Training process is little slower with variable sequence length as we can see in figure 5.9.

If we trained a neural network with variable length sequence for 500 epochs, training loss will drop to around 2.5 px. Behaviour of training loss and validation loss in time is visualised in the figure 5.10.

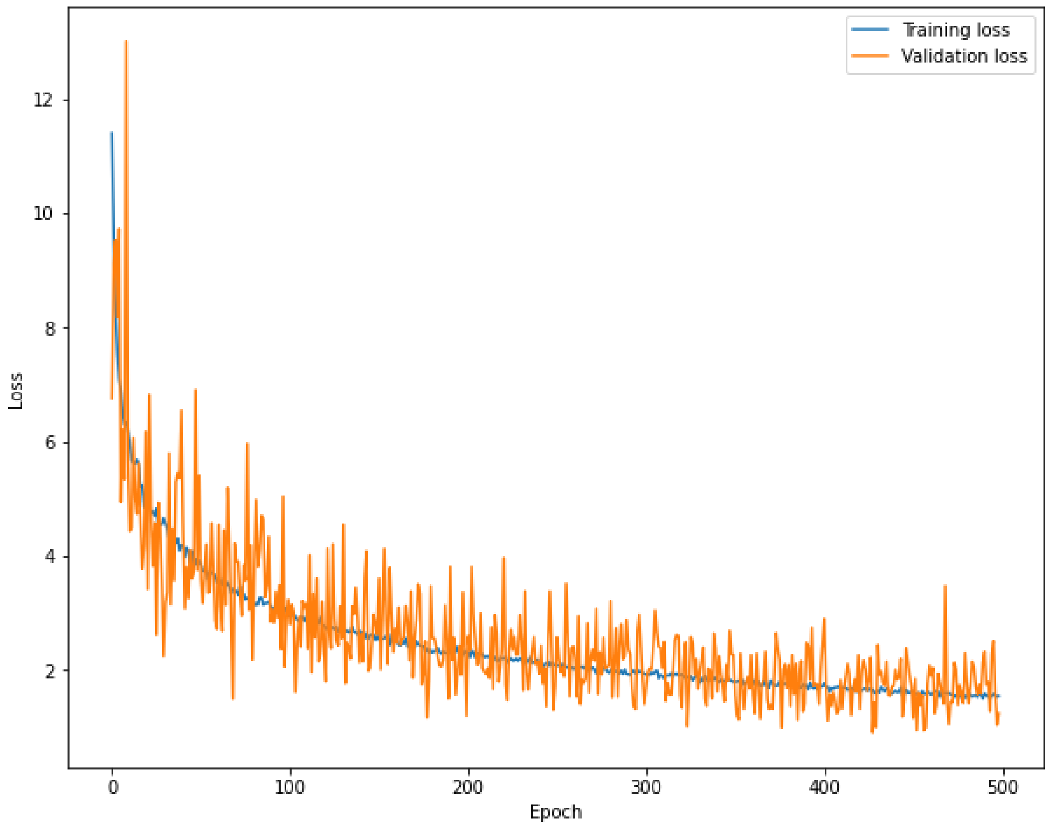


Figure 5.10: Training loss (in blue) and validation loss (in orange) during training for 500 epochs for model trained with variable length input.

This approach can perform quite well on inputs with length from 2 to 7 as in the table 5.7. Model loss increases with sequence length higher than 7, because model was not trained on sequences that long.

input sequence length	error in [px]
2	2.284
3	0.2325
4	0.181
5	0.15
6	0.1418
7	0.1525
8	0.4566
9	1.2356
10	2.401

Table 5.7: Testing error for input sequences with different length. Each length was tested on 100000 examples 5 times to get average error.

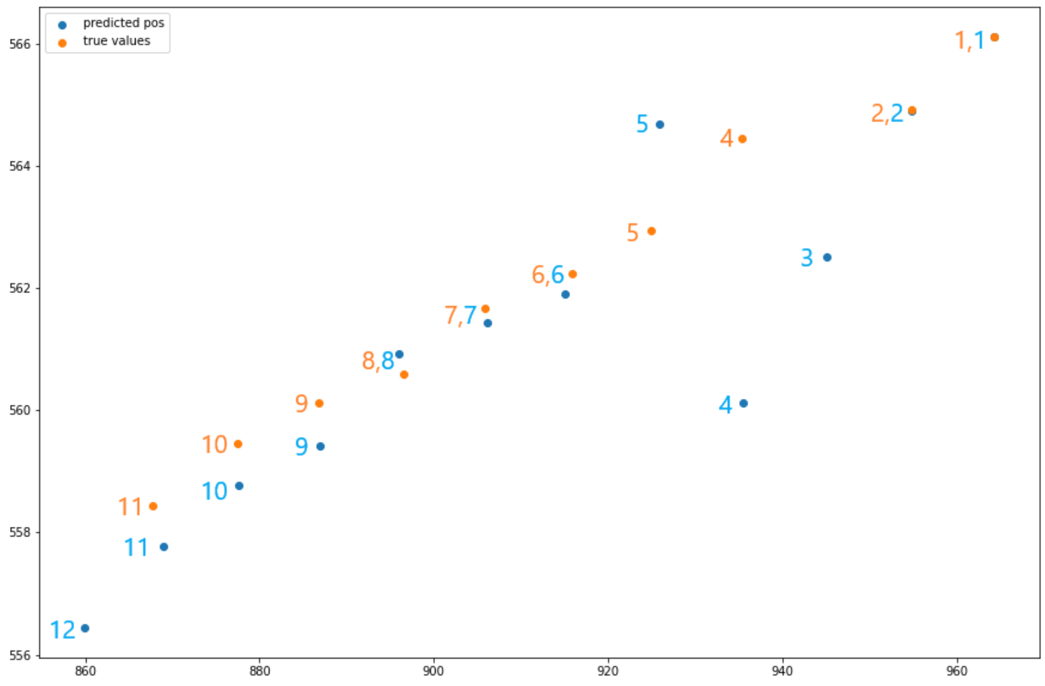


Figure 5.11: Positions of the moving object. Object is moving from top right corner to bottom left. Blue points are predicted positions by the model. Position prediction was made by model trained with variable input sequence length.

The figure 5.11 shows the model's performance on real data. Target sequence is the same as in the figure 5.8. From these graphs we can say that model with variable input sequence length is more consistent for long sequences. Predicted positions 8, 9, 10, 11 have same direction as real positions in contrast to the previous model where these positions did not lay on a straight line.

Advantages - Model is using all available information. Model has more consistent prediction precision.
Disadvantages - Longer training time.

5.3.4 Network output

The next position is predicted from previous object's positions using two possible approaches:

1. **Position prediction**
   Direct prediction of the next object position.

2. **Vector prediction**
   Direction vector prediction - direction from last position. The last position is shifted by direction vector to get the next position.

Vector vs. position prediction

The output of a model could be a **position** of an object in the picture or a **vector** representing direction from last position to new position. Performance of the model, with either of these output forms, is the same. The output form does not affect amount of information used for prediction. The vector output form is more practical for further operations.

5.3.5 Final Model

From these options, **variable input sequence length** and **vector output form** was chosen. Knowledge of the direction vector is needed during the processing. Thanks to the output form, it does not have to be computed from its positions.

5.4 Testing data

The Department of Astronomy and Astrophysics has provided 26 processed image series from AGO70 telescope. Each series contains positions of one of the 5 real moving objects. 24 series contain 20 images, one series of 19 and one series of 21 images. Testing data also contains verified ground-truth tracklets if not stated otherwise. The information about objects are from NY2o website [11].

1. **Navstar 51**
   - **Info**: GPS satellites
   - **Orbit**: MEO
   - **Minimal altitude**: 19871.4 km
   - **Period**: 11 hour and 57.6 minute
   - **Number of Series**: 4

2. **Navstar 60**
   - **Info**: GPS satellite
   - **Orbit**: MEO
   - **Minimal altitude**: 19846.4 km
   - **Period**: 11 hour and 57.9 minute
   - **Number of Series**: 3

3. **Cosmos 2434**
   - **Info**: Glonass satellite
   - **Orbit**: MEO
   - **Minimal altitude**: 19126.3 km
   - **Period**: 11 hour and 15.7 minute
   - **Number of Series**: 4

4. **Cosmos 2460**
   - **Info**: Glonass satellite
   - **Orbit**: MEO
   - **Minimal altitude**: 19129.9 km
   - **Period**: 11 hour and 15.7 minute
   - **Number of Series**: 4

5. **AMC-14**
   - **Info**: Broadcasting satellite
   - **Orbit**: GEO (central Africa)
   - **Minimal altitude**: 35620.0 km
   - **Period**: 23 hour and 56.1 minute
   - **Number of Series**: 11

5.5 Results

System was used to process sequences from section 5.4.

5.5.1 System behaviour

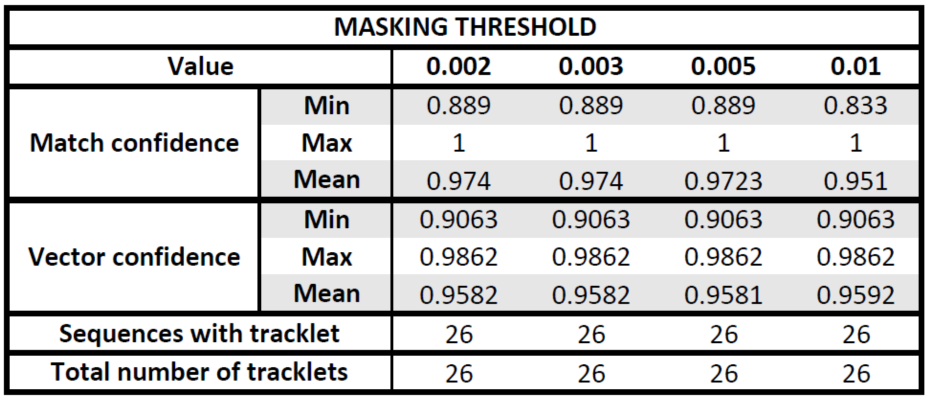
Sequences from section 5.4 were processed using our system. We tested behaviour of the system with different parameters. Default parameters are:

- masking threshold = 0.002
- matching threshold = 0.003
- K = 2
- overlap = 2

Output values, **match confidence** and **vector confidence** (section 4.1.2), show performance of the system. Match confidence shows length of the found tracklet relative to sequence length. Vector confidence shows how precise the model prediction was for given tracklet. Combination of these values can suggest if given tracklet represents the real moving object or random positions.

Masking threshold

Masking threshold is the maximum distance between positions in masking process (section 4.2).



MASKING THRESHOLD					
Value		**0.002**	**0.003**	**0.005**	**0.01**
Match confidence	**Min**	0.889	0.889	0.889	0.833
	Max	1	1	1	1
	Mean	0.974	0.974	0.9723	0.951
Vector confidence	**Min**	0.9063	0.9063	0.9063	0.9063
	Max	0.9862	0.9862	0.9862	0.9862
	Mean	0.9582	0.9582	0.9581	0.9592
Sequences with tracklet		26	26	26	26
Total number of tracklets		26	26	26	26

Table 5.8: Behaviour of the system with different values of masking threshold.

Total number of found tracklets and number of sequences with found tracklet is the same for all values.

The bigger the value is, the less positions are in the input sequence, so the system can process them faster. Yet, with high value we can lose some important positions from sequences. In the table 5.8, the minimal value of match confidence is the lowest for masking threshold 0.01. In this case, some positions which were in tracklet in previous runs were removed in masking process.

Matching threshold

Matching threshold is the maximum allowed distance between positions in matching process (section 4.5).

MATCHING THRESHOLD							
Value		**0.001**	**0.002**	**0.003**	**0.005**	**0.007**	**0.01**
Match confidence	**Min**	0.944	0.688	0.889	0.889	0.889	0.25
	Max	1	1	1	1	1	1
	Mean	0.964	0.946	0.974	0.979	0.98	0.81
Vector confidence	**Min**	0.968	0.913	0.9063	0.907	0.858	0.547
	Max	0.974	0.986	0.9862	0.986	0.986	0.986
	Mean	0.971	0.961	0.9582	0.961	0.957	0.854
Sequences with tracklet		3	21	26	24	25	26
Total number of tracklets		3	23	26	24	25	42

Table 5.9: Behaviour of the system with different values of matching threshold.

According to the table 5.9, matching threshold has a huge influence on system behaviour. As the value rises, so does the total number of found tracklets.

For value 0.01 the system found 42 tracklets, but many of them with relatively small confidence values. The minimal vector confidence of 0.58

tells us about model prediction precision. Decrease in mean vector confidence means many tracklets with low model prediction precision. The minimal match confidence value is 0.25 (tracklet length is 25% of sequence length).

K value

K value is described in the section 4.7.3.

parameter K						
Value		1	2	3	4	5
Match confidence	Min	0.944	0.889	0.188	0.167	0.118
	Max	1	1	1	1	1
	Mean	0.978	0.974	0.901	0.876	0.846
Vector confidence	Min	0.906	0.906	0.47	0.47	0.625
	Max	0.986	0.986	0.986	0.986	0.986
	Mean	0.956	0.958	0.935	0.934	0.944
Sequences with tracklet		25	26	26	26	26
Total number of tracklets		25	26	29	31	32

Table 5.10: Behaviour of the system with different values of K.

K value is responsible for the total number of found tracklets, according to the table 5.10. K value greater or equal to 3 causes the system to produce some tracklets with low confidence values, especially match confidence values. If K value is equal to 1, the system is not able to find tracklet for one sequence. Two images in this sequence do not contain moving object's position. Higher value of K ensures that the system will find missing tracklet.

Overlap

Overlap value is described in the section 4.7.4.

Overlap					
Value		2	3	4	5
Match confidence	Min	0.889	0.353	0.421	0.889
	Max	1	1	1	1
	Mean	0.974	0.933	0.954	0.974
Vector confidence	Min	0.906	0.898	0.843	0.91
	Max	0.986	2.205	0.989	0.989
	Mean	0.958	1	0.958	0.963
Sequences with tracklet		26	26	26	26
Total number of tracklets		26	28	27	26

Table 5.11: Behaviour of the system with different values of overlap.

Overlap does not have a huge effect on the system behaviour according to 5.11. There is one interesting anomaly for value 3. Maximum vector confidence is above 1. This means that one of found tracklets was describing stationary object with a small norm of direction vector. Formula for the vector confidence value is described in the section 4.1.2.

5.5.2 Analytic vs. RNN system

Tracklets, created by the existing analytic system, were compared with ones created by our new system using RNN with parameters: masking threshold = 0.002, matching threshold = 0.003, K = 2 and overlap = 2. RNN system was trained on artificial data. For every test sequence, tracklets found by both systems were the same - tracklets are the same or one is a part of the other.

In most cases tracklets found by the RNN system are longer, as shown in the table 5.12. In the figures 5.12, 5.13, 5.14 and 5.15, there are examples of output tracklets from both systems. Green points are positions for given measurement time that are in both tracklets, blue points are positions only from the RNN system tracklet and red points are positions only from the analytic system tracklet.

Difference In length	-1	0	1	2	3	4	5	6	7	8	9	10
Count	1	2	8	4	6	1	0	1	1	0	0	2

Table 5.12: Difference in number of points in tracklet between tracklets from the RNN system and the analytic system.

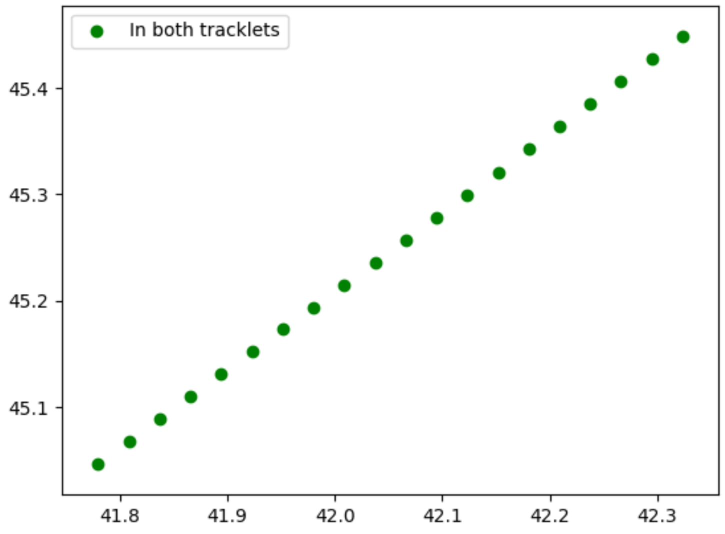


Figure 5.12: Tracklets for Cosmos 2460 satellite. Tracklets from both systems are the same.

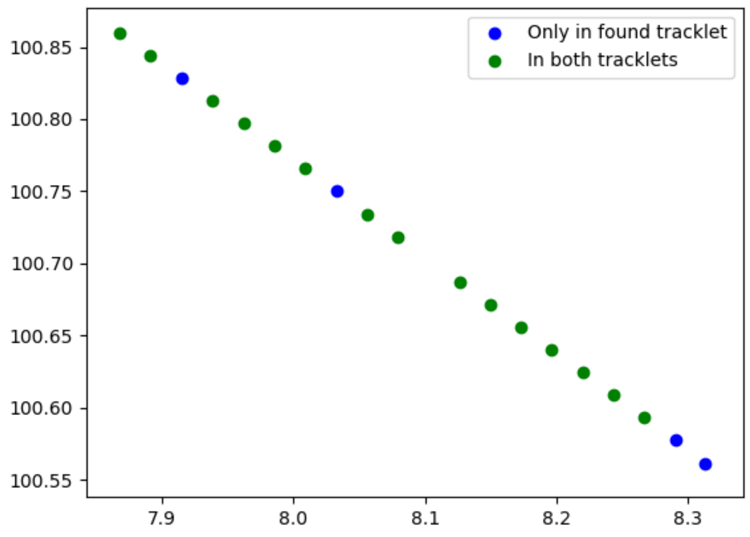


Figure 5.13: Tracklets for Navstar 51 satellite. Tracklet from the new system is 4 positions longer than tracklet from the analytic system.

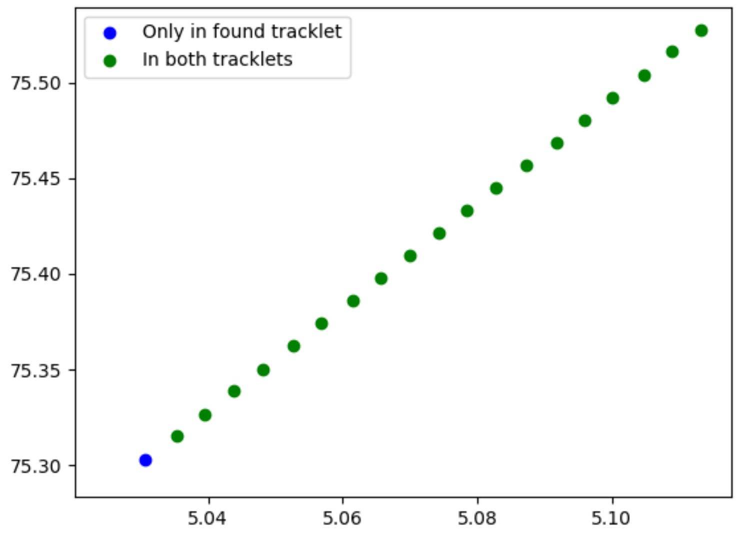


Figure 5.14: Tracklets for AMC-14 satellite. Tracklet from the new system is 1 positions longer than tracklet from the analytic system.

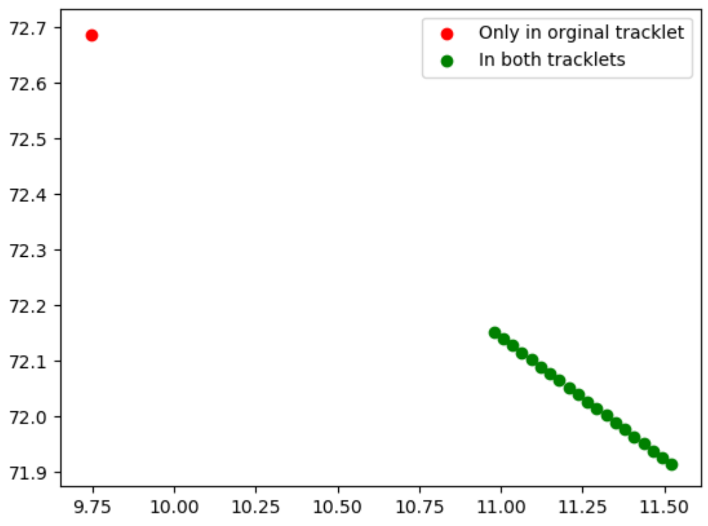


Figure 5.15: Tracklets for Navstar 60 satellite. Tracklet from the analytic system is 1 position longer than tracklet from the RNN system. The additional position from tracklet is a mistake made by the analytic system.

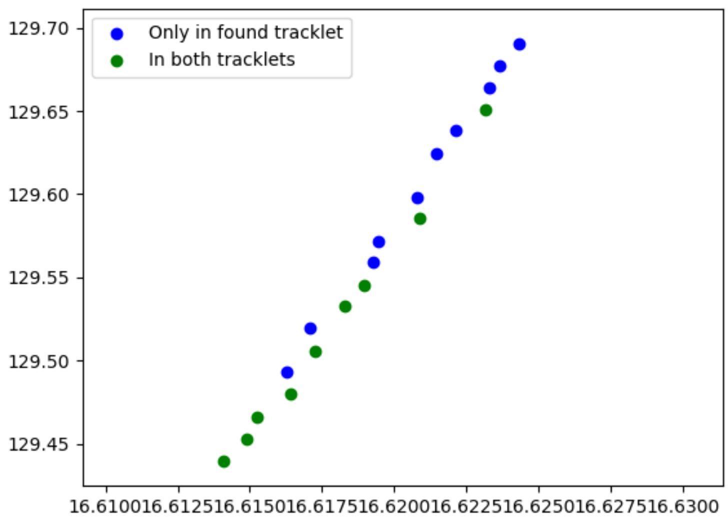


Figure 5.16: Tracklets for AMC-14 satellite. Tracklet from the new system is 10 positions longer than tracklet from the analytic system. Vector confidence value for this tracklet is 0.90.

Conclusion

We have shown that the RNN method is capable of finding moving object's positions in the sequence of images. Moreover, this method performs well on testing data from section the 5.4. In comparison to the existing analytical solution in the section 5.5, RNN method was able to find longer tracklets, which could lead to more precise identification of object's trajectory. Introduction of confidence values in the section 4.1.2 brings additional information about the found tracklet. Such information is missing in output of the existing analytical system. We have proved the concept of tracklet building as reliable using the RNN methods. In the future work, RNN method can be further improved to find moving objects with more complex trajectories, which will be a part of my PhD thesis.

Bibliography

[1] Jiri Silha, Stanislav Krajčovič, Matej Zigo, Danica Zilkova, Pavol Zigo, Jaroslav Simon, Juraj Toth, Leonard Kornos, Srinivas Setty, Tim Flohrer, et al. Development of the slovak 70-cm optical passive system dedicated to space debris tracking on leo to geo orbits. *Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS)*, page 85, 2019.

[2] Mgr. Stanislav Krajčovič. Development of an efficient tracklet building algorithm for space debris objects. Master's thesis, Comenius university in Bratislava Faculty of mathematics and informatics, 2018.

[3] The Europe space agency (ESA). Types of orbits. `https://www.esa.int/Enabling_Support/Space_Transportation/Types_of_orbits`, visited 27.1.2021.

[4] Stanislav Krajčovič, Roman Ďurikovič, and Jiří Šilha. Selected modules from the slovak image processing pipeline for space debris and near earth objects observations and research. In *2019 23rd International Conference Information Visualisation (IV)*, pages 112–117. IEEE-CS, 2019.

[5] Vladimir Kouprianov. Distinguishing features of ccd astrometry of faint geo objects. *Advances in Space Research*, 41(7):1029–1038, 2008.

[6] WT Thompson. Coordinate systems for solar image data. *Astronomy & Astrophysics*, 449(2):791–803, 2006.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Imran Ahmad. *40 Algorithms Every Programmer Should Know*. Packt Publishing, June 2020.

[9] Irtiza Hasan, Francesco Setti, Theodore Tsesmelis, Alessio Del Bue, Fabio Galasso, and Marco Cristani. Mx-lstm: Mixing tracklets and vislets to jointly forecast trajectories and head poses. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 10, 2018.

[10] S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi. Sequence-to-sequence prediction of vehicle trajectory via lstm encoder-decoder architecture. pages 1672–1678, June 2018.

[11] N2yo - live real time satellite tracking and predictions. `https://www.n2yo.com/`, visited 27.1.2021.

[12] Thomas Viehmann Eli Stevens, Luca Antiga. *Deep Learning with PyTorch*. Manning Publications Co., 2020.

[13] *PyTorch official website*, 2020. `https://pytorch.org`.