Worm Motion Analysis Program

This program is designed to track *C. elegans* nematodes as filmed through a microscope, and analyze their movement.

Table of contents

Written by Gady Brinker and Dror Feitelson for the Treinin lab, Hebrew University Hadassah Medical School, Jerusalem, Israel.

1. Installation and Setup

To install the software you only need to copy the executable.

Note that the software itself is only a small part of the disk space that you will use. There are also the movies and the results of the analysis. It may therefore be advisable to use a separate disk or partition for this project. In this manual we'll assume this is disk D. Alternatively, it can be a folder on the desktop, under "My Documents", or wherever.

It is suggested to create two folders:
- D:\movies
- D:\analysis

Copy the installation files into D:\analysis. This includes the following:
- The executable, which is called **bwstart.exe**. This is the only file that is really required.
- A sample parameters file, called **params.txt**. This is provided as an example that you can edit for your needs.
- This user manual.

Copy the movies you want to analyze into D:\movies.

System Requirements

This software was developed for a PC running Windows XP.

The movies and analysis require lots of memory. In our setup, a 1 minute movie occupies about 750 MB of disk space. The output includes a reduced-size copy of the movie with annotations, which requires additional disk space.

It is recommended to run this program on a system with plenty of RAM, say 1 GB or more.

2. Capturing a Movie

In principle this software should be able to work with movies captured in different settings. In practice it has only been tried with one specific setting.

The microscope was used with a lens that provides a magnification factor of 25.

The camera we use is a Nikon Coolpix

The camera is connected to a PC.

The movie is captured using labview. The actual rate of the movie is 10 frames per second (FPS), despite the data in the movie itself which says 30 FPS.

The movie is stored in avi format, with no compression. As a result the file is very big, typically on the order of 750MB.

Once you have a movie of a worm crawling across a Petri dish you are ready to analyze it using this program. Note, however, that developing this program required extensive tuning to get it to perform reasonably well. It may not work well for movies captured in other settings, especially if a different resolution is used.

3. Analyzing a Movie

Analyzing a movie requires a few steps. Initially this may look complicated, but it isn't really once you get the hang of it.

The steps involved are:
1. Setting the parameters
2. Executing the program
3. Interacting with the program to ensure that all is well

After you finish, you need to look at the result files that were created.

Setting the Parameters

Before you actually run the program, you need to specify parameter values. In most cases, just use the provided defaults. The only parameters you *must* specify are the folder where the movies reside and the name of the movie file.

The parameters file is a simple text file, called **params.txt**. To view and edit it just double-click on its icon. This will most probably open the Windows **notepad** editor.

The format of the file is as follows. Each parameter is specified in a separate line. The format of each line is

parameterName = parameterValue

An example of a parameters file is shown to the right:

```
moviedir      =  D:\\movies\\
moviename     =  e1.avi
frmps         =  10
mmpix         =  178
fframe        =  0
lframe        =  0
screenop      =  0
debugimg      =  0
seglen        =  0.5
pausespdlim   =  0.05
subbg         =  0
rsedg         =  16
objclniter    =  2
wsiblck       =  0.035
disjntdel     =  20
segangdif     =  60
```

The main parameters you may want or need to change at this stage are the following:

- **moviedir** – the folder where the movie file is located. Note the double slashes (\\) used to separate folders and subfolders, and that it should also end with a double slash.
- **moviename** – the name of the movie file.
- **frmps** – the speed of the movie, in frames per second. This is important for calculating the worm's speed correctly.
- **mmpix** – the movie's scale: how many pixels correspond to one millimeter. Again, this is important for calculating the worm's speed correctly. This may be measured by creating a short movie of a small ruler, and seeing how many millimeters is the full frame. (This assumes you know the frame size in pixels.)

- **fframe –** the first frame, where to start the analysis. Usually this is 0, meaning from the start of the movie.
- **lframe –** the last frame, where to stop the analysis. Usually this is also 0, meaning till the end of the movie.
  fframe and **lframe** are used to analyze only part of a movie. The most common scenario is when the worm simply crawls out of the frame, and you need to stop the analysis when it is still completely visible (the program tends to crash if only a small part of the worm is left). Another case is when there is a problematic point in the middle of the movie that the program can't handle, e.g. due to the worm making a loop that closes on itself. When this happens, you can analyze the part before the problem and the part after the problem separately.

Additional parameters that may be useful if you want to dig deeper include the following:

- **screenop –** if set to 1, details of the analysis will be displayed on screen during execution. 0 means not to show such details.
- **seglen –** the motion analysis is based on dividing the movie into short segments, and analyzing the motion in each such segment. Later, successive segments with consistent motion are united into longer segments. **seglen** is the length in seconds of the basic short segments. This requires **frmps** to be set correctly.
- **pausespdlim –** one of the motion types identified is a pause, where the worm does not move. This parameter sets the pause speed limit: if the motion is slower, it will be considered a pause. The units are millimeter per second. Correct calculation of the speed depends on correct setting of the **frmps** and **mmpix** parameters.
- **segangdif –** maximal angle between movement segments that still qualifies as the continuation of the same movement.

The other parameters are only used when you really want to manipulate how the program performs the analysis. Some are described in the "Troubleshooting" section below.

The program only reads lines with the above format. Lines that do not have the "=" in the middle are ignored. Thus you can add comments to the file and it should still work.

Executing the Program

To execute the program, simply double-click on its icon. This will cause a console window to open – a small black window where the program displays text output during its execution.

The program will read the parameters file, and start analyzing the movie. Note that the parameters file must reside in the directory (folder) where you run the program.

As the program runs, it will first print the name of the movie it is analyzing:

```
Analyzing  D:\\movies\\e1.avi
```

Then, after a pause, it will print the number of each frame it is working on:

```
#0  #1  #2  #3  #4  #5  #6  #7  #8  #9  #10 …
```

If the **screenop** parameter is set, more data will be printed out.

Checking that the Analysis is OK

The analysis actually involves the following steps:

1. Identifying the worm in each frame
2. Finding the worm's "skeleton" (a line that goes down its middle)
3. Identifying the worm's head and tail (that is, which end is which)
4. Calculating various structural metrics, such as how symmetric the worm is
5. Dividing the movie into segments of consistent movement
6. Calculating statistics of the movement

Some of these steps may occasionally fail, which may lead to erroneous results. The biggest problem is correct identification of the head and tail, especially after the worm forms a loop and then opens up again. The program solves this problem by asking you for your help. Specifically, it creates a set of files that represent frames of the movie, each showing the worm, its skeleton, and its head. You are supposed to verify that it is correct. A good tool to use for this is VirtualDub.

Installing and Using VirtualDub

VirtualDub is a free program for handling and creating movies on a Windows platform. We use it to collect the individual frames we create, view them, and potentially turn them into a movie.

VirtualDub is available from www.virtualdub.org. Download it from the downloads page (this is actually located on SourceForge). There is no complex installation process – just unzip the downloaded archive in a folder of your choice. A good place may be

```
C:\Program Files\VirtualDub\
```

Then create a shortcut to the executable **VirtualDub.exe** from the desktop or wherever is convenient.

VirtualDub is a graphical program. To execute it, double click on its icon or shortcut. Then click the **file** menu, and select **Open video file…** (which is the first option). This will open a file browsing window. Navigate to the analysis folder, then to the results subfolder named after the movie you are analyzing, and finally to a subfolder there named **frames**, e.g.

D:\analysis\e1.results\frames\

Make sure that the shown file types are either “all types” or “image sequence”. With these settings, you will see that this subfolder contains many files called frame0000, frame0001, frame0002, and so on. Select the file **frame0000** (or the lowest number there is, in case you didn’t start from 0) and click on **open**. VirtualDub will then import all the subsequent files as well. Initially, it will show the first frame twice: one is the input panel, and the other the output panel. As we are not going to perform any editing or manipulations, the output panel does not concern us.

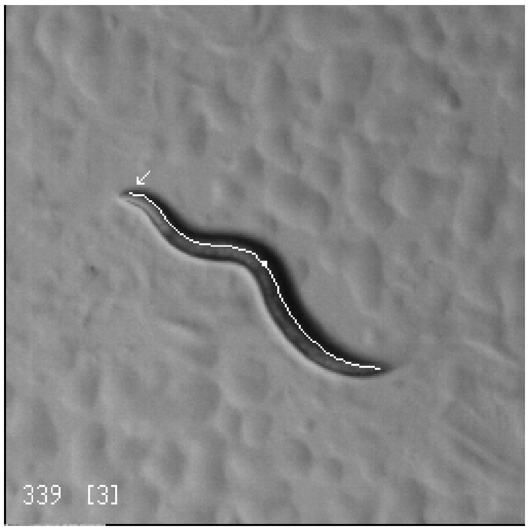
For our purposes, all we need is to view all the frames. Use either of the two “play” buttons at the bottom left ▷ . To ensure you don’t miss anything, you can stop the movie ▢ and then step one frame at a time using the buttons with the two triangles ▷▷ (which typically denote fast forward or fast backward on a VCR). You can also move the slider at the bottom of the window to scan the frames or reach whatever part of the movie you want to get to.

If you want to create a movie of these individual frames (which can then be viewed using Windows media player or other such programs), do the following:

1. From the **video** menu select **Frame Rate…** and set the correct frame rate for your movie.
2. Again from the **video** menu select **Compression…**. You may leave it as uncompressed (the first option), and skip this step entirely. However, the resulting movie file will be very big. We found that using **cinepak** (the second option) reduces the movie size by about 90%, with only a very small degradation in quality.
3. From the **file** menu, select **Save as AVI…**. This opens a file browsing window to allow you to specify the file name to use.

Verifying Head-Tail Identification

The program partitions the movie into several segments in which the head-tail identification seems consistent. In some cases, the whole movie may be a single segment. It then asks for your approval regarding the identification in each segment. This is done by marking the head with a small arrow in each frame, thus:



The number at the bottom left is the frame number. If this is followed by an **x**, it means the frame is considered “inconsistent”. This means that the worm’s skeleton length in this frame deviates significantly from the average skeleton length. “Significantly” means by more than about 10%. This typically happens due to problems with identifying the extent of the head.
Inconsistency is just a warning that something may be wrong, but it does not affect the rest of the analysis. It could add noise to the results, because movement is measured by the worm’s midpoint, and if we fail to identify the head correctly in one frame but not in another, the midpoint moves even if the worm actually didn’t.

The number after the frame number, in square brackets, is the segment number. The boundaries between segments are typically frames in which the program failed to identify the worm’s skeleton, and therefore also couldn’t identify its head and tail.

You need to go over all the frames in the segment, and check the head identification. If the arrow indeed always points to the head, answer **y** (yes). If it points to the tail, answer **n** (no). In this case, the program will switch the identification for all the frames in this segment of the movie.

If you observe inconsistent identification within a segment, the program has failed. The analysis will then be unreliable, at least regarding metrics that concern forward or backward movement. While possible in principle, we didn’t actually see such failures.

After you verify the head/tail identification in all the segments, exit VirtualDub or whatever other application you used to view the frames. The program waits for you to do so. When ready, hit enter to tell the program is can proceed. This step is needed to allow the program to overwrite the frames with the final movement annotations. If you do not exit VirtualDub, windows may not allow them to be overwritten.

Frames that Fail

In some cases, the program may fail to identify the skeleton in a certain frame. That frame will thereafter be dropped from the analysis.

The most common reason for failing to form a skeleton is if the worm made a loop that closes on itself. Other problems are when the analysis erroneously creates a skeleton with a fork or a gap.

Failed frames are shown in the sequence of frames used to check head-tail identification, but without a skeleton, and with the label “failed”.

The badly-formed skeleton that was extracted from such frames is stored in a subdirectory called **fail**. This may be useful for debugging.

4. Reading the Output of the Analysis

The output of the analysis is contained in several text files and some picture files. All these files are located in the results folder named after the movie, under the analysis folder. In our example this would be

D:\analysis\e1.results\

The files are

- **prm.txt** – a copy of the parameters used in the analysis, for the record. This file can also be used as the parameters file for a new analysis, to check that the results are reproduced.
- **wxs.txt** – output concerning worm movement metrics, as described below.
- **wxo.txt** – output concerning worm shape metrics, as described below.
- **wxd**.txt – a dump of the raw data used to create wxo.txt.
- **spd.txt** – output giving the speed in each frame.
- **track.bmp** – a picture of the worm's track throughout the movie, as described below.
- **frames\frameXXXX.bmp** – annotated frames of the movie (where XXXX is the frame number). Each frame is annotated with the movement type that was taking place, as described below. In addition, the first backwards movement frame that occurs after a forward movement is labeled as a *reversal*. You can use VirtualDub to turn these frames into an annotated movie, as described above.

The wxs.txt File

This file contains data about the worm's movement throughput the movie.

The first part of the file is a listing of the movement segments identified in the movie. As explained above, the program works by first dividing the movie into short segments of length **seglen** seconds (this is a parameter; the default is seglen=0.5). It then classifies these segments into four types:

- *Forward movement*. This is identified by computing the distance from the worm's midpoint to its head in the first frame, and comparing it with the distance from the worm's midpoint in the *last* frame to the head in the first frame. If the midpoint moved towards the head, this is forward movement.
- *Backward movement*: if the midpoint moved away from the head.
- *Pause*. This is defined as movement slower than **pausespdlim**, which is a parameter (default of 0.05 mm/sec). Note that movement is measured at the midpoint. Thus the head can perform foraging movements but the segment will still be classified as a pause. This can be verified by looking at the midpoint of the skeleton, which is specially marked in the output frames.
- *Omega*. This label is used if three shape-related conditions are met:
    - There are only two cutpoints, i.e. the worm's skeleton does not intersect the straight line from its head to its tail, and

    - Its amplitude (the maximal distance from the head-tail line to any point on the skeleton) is at least 40% of the skeleton length, and
    - The average distance of skeleton points from the head-tail line is at least half the maximal amplitude.

  Note that “omega” is contagious – if any frame in the segment is an omega, the whole segment is classified as omega.

After labeling the short segments, the program attempts to unify adjacent segments into longer segments. This is done according to the following rules:

- Adjacent movement segments are unified if they are both in the same direction (that is, both are forward or both are backward), and if the angle of movement has not changed too much. The maximal angle change within the same segment is given by the **segangdif** parameter. When many short segments are being united one after the other, the angle is measured between the direction of the initial short segment and the last one being considered.
- Adjacent pause segments are unified.
- Adjacent omega segments are unified.

After the list of movement segments some statistics of the movement are given (average forward speed, average backward speed, percent time moving forward, moving backward, or pausing, etc.). These are self explanatory.

The last metric listed is the **roam ratio** (or, rather, a better name may be the **dwell ratio**). This attempts to quantify the degree to which the worm moved consistently in the same direction and covered some distance, as opposed to moving back and forth in the same area or pausing a lot. It is calculated by the following formula:

$$\text{roam\_ratio} = 1 - \frac{\text{end\_to\_end\_dist}}{\text{accum\_dist}}$$

where

- **end_to_end_dist** is the distance along a straight line from the midpoint in the first frame of the movie to the midpoint in the last frame of the movie.
- **accum_dist** is the cumulative distance covered in all the movement segments in the movie (that is, in all segments of length **seglen** that were not labeled as a pause). The distance in each segment is measured along a straight line from the skeleton midpoint in the first frame of the segment to the skeleton midpoint in the last frame of the segment, and these are summed to get the cumulative distance.

Thus if the end_to_end_dist is very small relative to the accum_dist, meaning that the worm moved a lot but essentially stayed in the same place, the roam_ratio will be close to 1. If the worm moved consistently and managed to cover considerable distance, the end_to_end_dist will be close to the accum_dist, and the roam_ratio will be low. Thus the roam ratio actually measures the degree to which the worm tends to dwell in the same place. However, it will never reach 0 because accum_dist accounts for the sinusoidal pattern of the midpoint’s movement, and will always be bigger than the end_to_end_dist. Somewhat counterintuitively, the roam ratio will also be low for worms that hardly move at all, because then both the end_to_end_dist and the accum_dist are small.

Note that the roam ratio depends on the movie length, and it is dangerous to compare roam ratios from movies with different lengths (which reflect movements for different durations).

The wxo.txt File

This file contains data about the worm's shape and how it changed throughout the movie.

A slew of metrics are evaluated. Each is measured separately for the different types of movement (moving forward, moving backward, pausing, or looping to form an omega shape) that were defined above, and also for the whole movie together. Movement is measured at the worm's midpoint, so foraging movements of the head do not count. Pauses are defined to be movement of up to **pausespdlim** mm/sec, which is a definable parameter, as measured over a duration of **seglen** (another parameter).

For each metric/movement combination, the program calculates the average of the metric over those segments of the movie that exhibit this type of movement. It also characterizes the distribution of metric values, by providing deciles: the minimum value, the 10th, 20th, 30th, 40th, …, 90th percentiles, and the maximum.

The metrics are the following.

numsklpts is the number of pixels in the skeleton. It thus depends on the resolution of the movie.

len is the length of the worm in pixels. The difference from **numsklpts** is that **len** takes into account the actual distances involved, and doesn't just count pixels (pixels along a diagonal represent a larger distance than pixels along a horizontal line). It is calculated as follows:

- Divide the skeleton into 20 parts with the same number of pixels in each.
- Calculate the length of each part as the distance from the first pixel to the last one, using Pythagoras's theorem, and sum these distances
- If any pixels were left over in the first step, add this number of pixels.

cutpnums is the number of cutpoints. This is the number of times the worm's skeleton cuts a straight line from its nose to its tail, including the end points. As pixels are discrete, "cuts" means a distance of less than a pixel. A sequence of 3 or more skeleton points at a distance of more than 1.4 pixels is required between successive cutpoints.

avgamp is the average amplitude, in pixels. This is the average distance of the **numsklpts** points on the skeleton from the straight line connecting the nose to the tail.

ampsym is a metric for symmetry that is related to **avgamp**. It is the average of the *signed* amplitudes of skeleton points, i.e. their distances from the straight line connecting the head to the tail where a distance to one side of this line is considered positive and a distance to the other side negative. A value near 0 implies symmetry.

maxampL and **maxampR** are the maximal amplitudes to the left and to the right. This is the maximal distance of any point on the skeleton from the straight line connecting the nose to the tail. "Left" and "right" are relative to the direction from the nose towards the tail.

Navgamp is the average amplitude as above, divided by the skeleton length (**len**).

NmaxampL and **NmaxampR** are the maximal amplitudes to the left and to the right as above, divided by the skeleton length (**len**).

avgangle is the average angle, which is a measure of the curvature along the worm. Angles are measured from the head towards the tail, and are in the range of ±180°, indicating an inclination to the left (positive) or to the right (negative).
Angles are measured by defining a step size that is one twelfth of the skeleton length (in pixels). Consider three skeleton points *a*, *b*, and *c*, that are one step apart. The angle at point *b* is the angle between the line from *a* to *b* and the line from *b* to *c*. Thus if the worm is completely straight, the angle will be 0.
The average angle is the average of the absolute value of the angle as measured for *all* skeleton points (except near the ends). Due to using the absolute value, positive and negative angles (inclination to left or right) do *not* cancel out.

anglesym is another metric symmetry, this time related to **avgangle**. It is based on averaging the actual values, which may be positive or negative depending on whether the worm tends to the left or the right at each point. Thus a value near 0 implies symmetry. If the worm does an omega-like shape all the angles will be in the same direction, leading to a large (positive or negative) value.

maxangL and **maxangR** are the maximal positive and negative angles, respectively.

area is the number of pixels covered by the worm. **thickness** is the average "width" of the worm in pixels: it is the area divided by the length of the skeleton (**len**). Both these metrics depend on the resolution.

straightfs is an attempt to quantify how straight the worm is. It is a normalized metric in the range [0..1], with values above ½ indicating that the worm is "straight".
The original metric was calculated as $1-\sqrt{(\text{stamp}^2 + \text{stang}^2)}$, where stamp = max(0.707, 1.5***maxamp**), and stang = max(0.707, **avgangle**/40). The limit to 0.707 is designed to limit each term to ½ when squared, so that their sum will be limited to 1.
Currently this metric is calculated in a simpler manner, which divides the distance from the nose to the tail by the actual length of the worm (**len**). A value near 1 thus indicates straightness, whereas a value near 0 indicates a loop.

sinusfs is an attempt to quantify how "sinus-like" the worm is, but actually it is more of a metric of symmetry. It is also normalized to [0..1], with values above 0.8 considered "sinus like". It uses the symmetry metrics defined above, with the notion that symmetry implies sinusoidal shape. It is calculated as $1-\sqrt{(sinamp^2 + sinang^2)/2}$, where sinamp = **ampsym** / **avgamp**, and sinang = **anglesym** / **avgangle**. Thus if the amplitudes and angles tend to cancel out, we will get a value near 1. If they do not, the value will be low.

The wxd.txt File

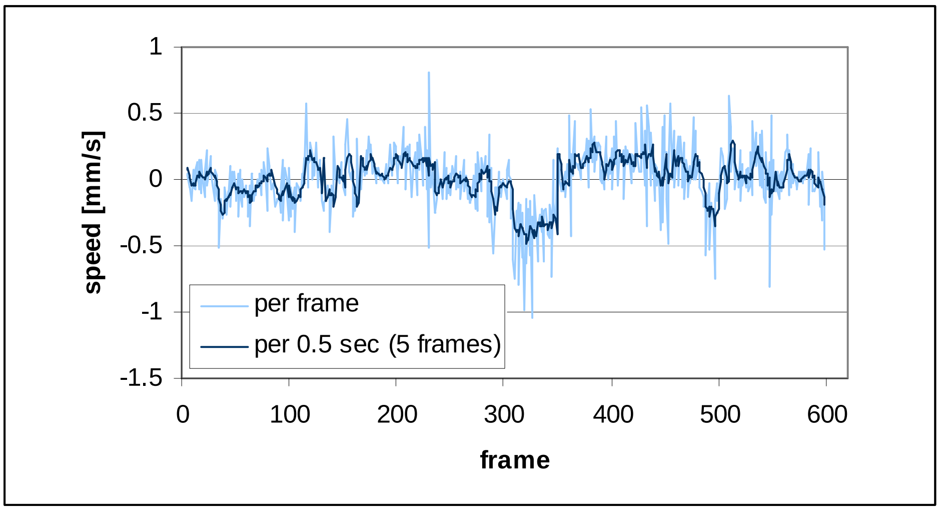
This file contains all the above data fields for every frame of the movie. The fields are separated by commas.

The spd.txt File

This file provides data about the momentary speed of the worm in each frame. Positive values indicate forward movement, and negative values indicate backwards movement. The classification of movement segments is also indicated.

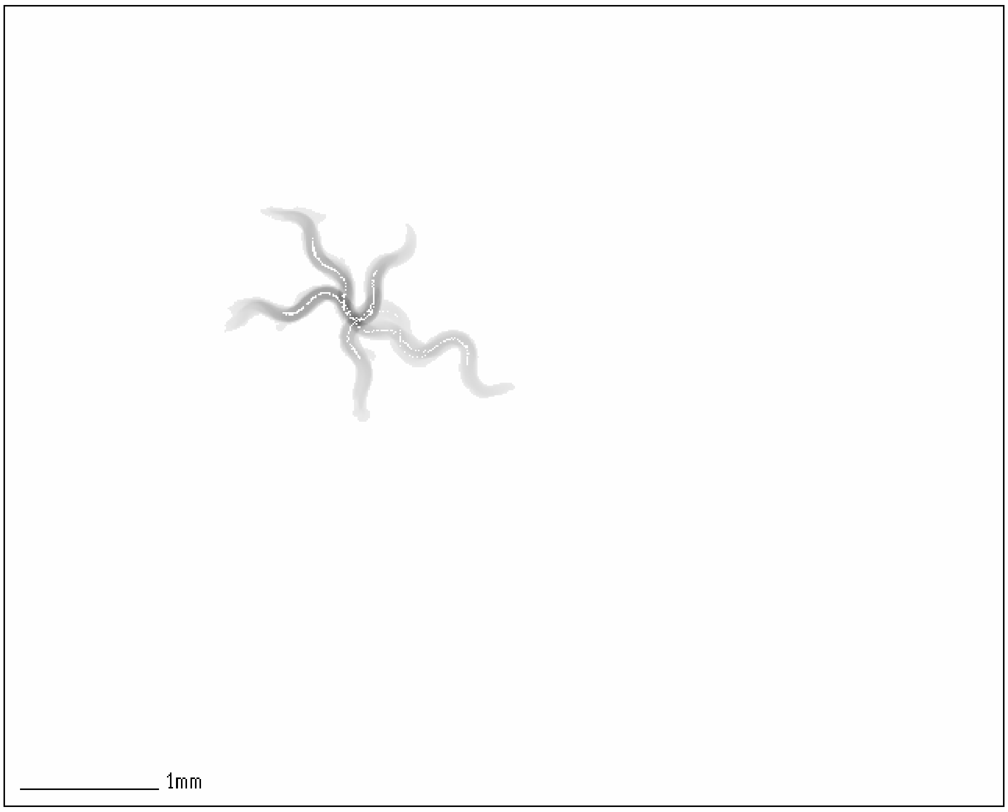
Two values are given for each frame. The first is the speed as calculated between this frame and the previous frame. The second is the speed as calculated between this frame and the frame half a second ago (for example, for movies captured at 10 frames per second, this would be 5 frames back). In either case, the speed is calculated as the distance traveled by the midpoint divided by the time span. The units are millimeter per second, and depend on correct setting of the **mmpix** and **frmps** parameters.

The data is tab-delimited. It can be loaded into Excel and plotted (or plotted using some other program). A typical output is as follows:



The track.bmp File

This picture gives a visual summary of the worm's movement throughput the movie. A typical picture (of a worm that mostly moved back and forth in the same place) looks like this:



The shading shows the worm's movements. Areas where the worm paused for a long time, or visited repeatedly, will have a darker shading. Areas that were only passed through once will have a light shading.

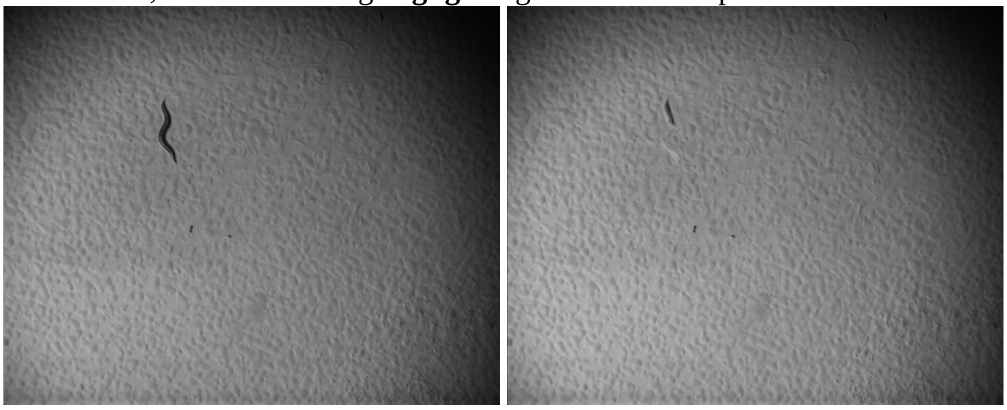
The white dots within the shading show the locations of the worm's midpoint in the different frames of the movie.

The scale bar shows one millimeter. Its accuracy depends on the **mmpix** parameter.

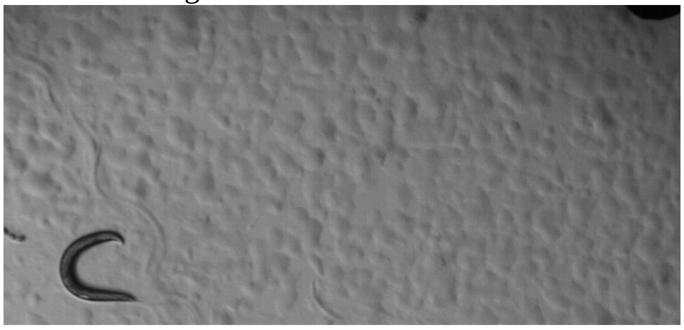
5. Troubleshooting

In some cases a problem is self evident, e.g. when the program crashes. But there are other problem situations that require attention too. For example, if you get lots of frames that failed skeleton identification, this is a problem that may be fixed by setting different parameter values in the parameters file.

In case of trouble, you can get additional data regarding what the program is doing. Two parameters can be used:

- **screenop** – if set to 1, the program will print out detailed information about its progress.
- **debugimg** – during the main loop of processing the movie's frames, create output files representing different steps in the processing. The main files are:
  - **thisbmp, resized** – the current frame in original and smaller size
  - **bmpWObg** – frame after background subtraction, if **subbg** is set to 1.
  - **avgbg** – the background that is being subtracted (again, if **subbg** is 1). This should be what you intuitively expect to be the background. It is identified by virtue of not changing throughout the movie. If any part of the worm is clearly visible, it means that the worm didn't move enough to be recognized as non-background. In that case, **subbg** should be set to 0 and **rsedg** adjusted, as described below.
    The following is an example of a frame from a movie where the worm doesn't move much, and the resulting **avgbg** image that includes part of it:
    
    Another possible problem is that the worm's movement causes small ripples on the surface, which are brighter than the normal background. If this happens, the identified background will include a bright tracking of the worm, and after subtracting it the frames will be polluted with this tracking, as exemplified below. Again, the solution may be to set **subbg** to 0.

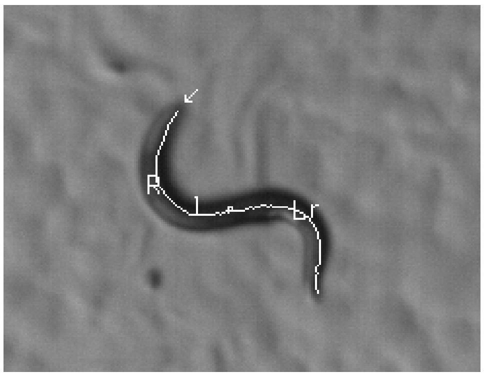
- **edge0, edge** – results of performing edge detection. This should clearly indicate the outline of the worm. If this isn't the case, adjust **rsedg** as explained below.
- **wormarea** – the area identified as containing the worm. If the worm does not appear, appears only partly, or has very large margins in some directions, this means that the identification did not succeed. An example of a problem situation is the following:



- **worms00** to **worms05** – different stages of creating the worm skeleton.

Note that these files represent the current frame being analyzed. As we progress from one frame to the next, the files are overwritten with new images. If the program crashes, they may contain images for the last two frames: some that have been overwritten already with current frame data, and some that were left over from the previous frame.

When **debugimg** is set to 1, this also causes additional data to be marked on the individual frame images in the **frames** directory. Specifically, the following markings are added:



- **R** marks the point with the maximal amplitude to the right, relative to the direction from the note to the tail. If the whole worm is to the left of the nose-tail line, there will be no R mark. This happens, e.g., when the worm forms a loop.
- **L** marks the point with the maximal amplitude to the left, in analogy to the above.
- **r** marks the point with the maximal

angle to the right.
  - **l** marks the point with the maximal angle to the left.

  As may be expected, **R** and **l** tend to go together, as do **L** and **r**.
- **exitonfail –** if set to 1, the program will exit if it fails to create a skeleton in some frame.

The parameters file also has several parameters that may be modified in order to improve the program's effectiveness for different movies. These are:

- **subbg** – subtract the background, to make the worm stand out better. By default, this is turned off (value of 0). Subtracting the background is required in order to avoid focusing on sharp distractions, such as the edge of the plate, bubbles in the substrate, or eggs.
  The background is defined to be the brightest grey level at each pixel taken over the whole movie. This usually works fine; it will only fail if the worm is stationary and stays in the same place for the duration of the movie. If this is a problem, turn background subtraction off by setting its value to 0. Note that you will probably need to reset **rsedg** as well, as described below.
- **objclniter –** number of cleaning iterations to perform. The goal of cleaning iterations is to get rid of spurious noise in the image. Each iteration cleans some of the smaller elements, assuming that the worm is the biggest.
- **rsedg –** threshold used in edge detection: pixels where the horizontal or vertical change is bigger than this are considered an edge. If the worm's outline (as shown in the **edge** debug image) is broken or partial, or the enclosing rectangle (as shown in the **wormarea** debug image) is too small, cutting off parts of the worm, you may need to lower **rsedg** to find the missing edges. If the area is too big (as in the example shown above), you have found spurious edges, and need to increase **rsedg** to avoid them.
  Note that **rsedg** interacts with **subbg**. When **subbg** is used, the contrast is increased, and a higher value should be used for **rsedg** (default of 25). If **subbg** is not used, **rsedg** should be decreased (default of 16).
- **wsiblck –** this parameter controls the expected area (as measured by the number of pixels) of the worm itself. It may depend on the resolution of the movie.
  If the worm as shown in **worm1** has holes in it, you need to enlarge **wsiblck** to enable more pixels to be associated with the worm. On the other hand, if the worm seems to include non-worm patches, make **wsiblck** smaller.
  If **wsiblck** is set to 0, the program tries to find the threshold automatically.
- **disjntdel –** size of disjoint elements (noise) to delete.