Module Interface Specification for Image Feature Correspondences for Camera Calibration

Kiran Singh

March 18, 2025

1 Revision History

Date	Version	Notes
2025-03-19	1.0	Initial Release

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/KiranSingh15/CAS-741-Image-Correspondences/blob/main/docs/SRS/SRS.pdf.

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by the Image Feature Correspondences for Camera Calibrationsoftware.

Data Type	Notation	Description
character	char	a single symbol or digit
string	str	a sequence of characters
boolean	$\mathbb{F}_2$	a number in the binary field, where all elements are {0,1}
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of Image Feature Correspondences for Camera Calibration uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Image Feature Correspondences for Camera Calibration uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	Input Parameters Input Format Module Specification Parameters Output Format Module Output Verification Module Control Module Image Smoothing Module Keypoint Detection Module Feature Descriptor Module Feature Matching Module
Software Decision	Sequence Data Structure Image Data Structure Module Image Plot Module Feature Match Data Module Dataframe Structure Module ORB Data Structure Module

Table 1: Module Hierarchy

6 MIS of Input Format Module

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

6.1 Module

config

6.2 Uses

- specParams (Section 7)

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_head_directory	-	head_path as string	noHeadFound
get_active_functions	-	tuple (user-methods)	-
get_chosen_parameters	-	tuple (user-params)	-
get_img_names	head_path as str	img_names as str^n	-
check_limits	tuple (user-params)	-	badKernelSize, badStdDeviation, badFASTThrehold, badBinSize, badPatchSize

6.4 Semantics

6.4.1 State Variables

- kernel_sz $\in \mathbb{Z}$
- std_deviation $\in \mathbb{R}$
- FAST_threshold $\in \mathbb{Z}$
- bin_sz $\in \mathbb{Z}$
- patch_sz $\in \mathbb{Z}$
- mthd_img_smoothing $\in \mathbb{Z}$
- mthd_kp_detection $\in \mathbb{Z}$

- mthd_kp_description $\in \mathbb{Z}$
- mthd_ft_match $\in \mathbb{Z}$

tuple of methods and parameters goes here.
set the state as the defaults,
then set the state as the user defined methods, if available

6.4.2 Environment Variables

- head_path as str

6.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

6.4.4 Access Routine Semantics

get_head_directory():

- output: head_path = Path(os.getcwd()) where head_path is a string

get_active_functions():

- output: [mthd_img_smoothing, mthd_kp_detection, mthd_kp_description, mthd_ft_match $\in \mathbb{Z}$]

get_chosen_parameters():

- output: [kernel_sz, bin_sz, patch_sz, FAST_threshold $\in \mathbb{Z}$, std_deviation $\in \mathbb{R}$]

get_img_names(head_path as str):

img_path = Path(head_path + "Raw_Images")
img_dir = Path(img_path)
input_img = [(file.stem, file.suffix, file.name) for file in img_dir.iterdir() if file.is_file()]
num_images = len(input_img)

- output: input_img $\in str^n$, num_images $\in \mathbb{N}$
- exception: none

check_limits():

- output: none

- exception: exc:=

$\neg(kernel\_sz < 1)$	$\Rightarrow$ badKernelSize
$\neg(kernel\_sz > 15)$	$\Rightarrow$ badKernelSize
$\neg(kernel\_sz \,\%\, 2 \neq 0)$	$\Rightarrow$ badKernelSize
$\neg(0 < std\_deviation < 10)$	$\Rightarrow$ badStdDeviation
$\neg(2 \leq FAST\_threshold \leq 255)$	$\Rightarrow$ badFASTThreshold
$\neg(1 \leq bin\_sz \leq 2048)$	$\Rightarrow$ badBinSize
$\neg(5 \leq patch\_sz \leq 100)$	$\Rightarrow$ badPatchSize

7 MIS of Specification Parameters Module

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

7.1 Module

specParams (Section 6)

7.2 Uses

None.

7.3 Syntax

7.3.1 Exported Constants

- $kernel\_sz := 5$
- $std\_deviation := 1$
- $FAST\_threshold := 15$
- $bin\_sz := 2000$
- $patch\_sz := 31$
- $mthd\_img\_smoothing := 1$
- $mthd\_kp\_detection := 1$
- $mthd\_kp\_description := 1$
- $mthd\_ft\_match := 1$

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_default_parameters	-	$kernel\_sz : \mathbb{Z}$ $std\_deviation : \mathbb{R}$ $FAST\_threshold : \mathbb{Z}$ $bin\_sz : \mathbb{Z}$ $patch\_sz : \mathbb{Z}$	-
get_default_methods	-	$mthd\_img\_smoothing : \mathbb{Z}$ $mthd\_kp\_detection : \mathbb{Z}$ $mthd\_kp\_description : \mathbb{Z}$ $mthd\_ft\_match : \mathbb{Z}$	-

7.4 Semantics

7.4.1 State Variables

$kernel\_sz : \mathbb{Z}$
$std\_deviation : \mathbb{R}$
$FAST\_threshold : \mathbb{R}$
$bin\_sz : \mathbb{Z}$
$patch\_sz : \mathbb{Z}$
$mthd\_img\_smoothing : \mathbb{Z}$
$mthd\_kp\_detection : \mathbb{Z}$
$mthd\_kp\_description : \mathbb{Z}$
$mthd\_ft\_match : \mathbb{Z}$

7.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

7.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

7.4.4 Access Routine Semantics

get_default_parameters():

- output:
  - $kernel\_sz : \mathbb{Z}$
  - $std\_deviation : \mathbb{R}$
  - $FAST\_threshold : \mathbb{Z}$
  - $bin\_sz : \mathbb{Z}$
  - $patch\_sz : \mathbb{Z}$
- exception: none

get_default_methods():

- output:
  - $mthd\_img\_smoothing : \mathbb{Z}$
  - $mthd\_kp\_detection : \mathbb{Z}$
  - $mthd\_kp\_description : \mathbb{Z}$
  - $mthd\_ft\_match : \mathbb{Z}$
- exception: none

8 MIS of Output Format Module

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

8.1 Module

formatOutput

8.2 Uses

- matchStruct (Section 10)
- dataframeStruct (Section 19)

8.3 Syntax

8.3.1 Exported Constants

Not applicable.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
-	-	-	-

8.4 Semantics

8.4.1 State Variables

-

8.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

8.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

8.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

8.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

9 MIS of Output Verification Module

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

9.1 Module

verifyOutput

9.2 Uses

None.

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

9.4 Semantics

9.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

9.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

9.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

9.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

9.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

10 MIS of Control Module

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

10.1 Module

main

10.2 Uses

- matchFeatures (Section 14)
- plotImage (Section 16)
- formatOutput (Section 8)
- verifyOutput (Section 9)

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	-

10.4 Semantics

10.4.1 State Variables

- kernel_sz $\in \mathbb{Z}$
- std_deviation $\in \mathbb{R}$
- FAST_threshold $\in \mathbb{Z}$
- bin_sz $\in \mathbb{Z}$
- patch_sz $\in \mathbb{Z}$

- mthd_img_smoothing $\in \mathbb{Z}$
- mthd_kp_detection $\in \mathbb{Z}$
- mthd_kp_description $\in \mathbb{Z}$
- mthd_ft_match $\in \mathbb{Z}$
- img_obj_1, img_obj_2 $\in \mathbb{Z}^{h \times w}$

10.4.2 Environment Variables

- head_dir as **str**
- path_input_img as **str**
- path_keypoints as **str**
- path_descriptors as **str**
- path_feature_matches as **str**

10.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

10.4.4 Access Routine Semantics

main():

- transition: Modify the state of the Specification Parameters Module and the environment variables for the Image Plot Module and Output Format Module.

[head_dir as str] = get_head_directory()

[mthd_img_smoothing $\in \mathbb{Z}$, mthd_kp_detection $\in \mathbb{Z}$, mthd_kp_descriptors $\in \mathbb{Z}$, mthd_ft_matching $\in \mathbb{Z}$] = get_chosen_methods()

[kern_sz $\in \mathbb{Z}$, std_deviation $\in \mathbb{R}$, FAST_threshold $\in \mathbb{Z}$, bin_sz $\in \mathbb{Z}$, patch_sz $\in \mathbb{Z}$] = get_chosen_parameters()

For each image, i
 # Smooth the image as a preprocessing step to keypoint detection
img_obj_1 = smooth_image(img_obj_1$\in \mathbb{Z}^{h \times w}$, kernel_sz$\in \mathbb{Z}$, std_deviation$\in \mathbb{R}$)

 # Identify the keypoints. Note that if the methods for keypoint detection and descriptors

are both == 1, then ORB is the selected method, and the keypoint and descriptor modules should use the same ORB object, which likely will come from the OpenCV library

 # Assign descriptors to keypoints

 # export keypoints to csv
 # export descriptors to csv

 # generate and save image with keypoints
 # generate and save image with scaled keypoints
##

 # Compare features between differing images

 # verify that the match structure conforms to the conditions in the Output Verification Module

 # export matche tuples to csv
 # generate and save images with corresponding matches

11 MIS of Image Smoothing Module

[You can reference SRS labels, such as R**??**. —SS] [It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

11.1 Module

smoothImage

11.2 Uses

- config (Section 10)
- imageStruct (Section 15)

11.3 Syntax

11.3.1 Exported Constants

None.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
smooth_image	noisy_img: $\mathbb{Z}^{H\times W}$, kernel_sz: $\mathbb{Z}$ std_deviation: $\mathbb{R}$	smoothed_img: $\mathbb{Z}^{H\times W}$	-

11.4 Semantics

11.4.1 State Variables

- smoothed_img: $\mathbb{Z}^{H\times W}$

11.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

11.4.3 Assumptions

- Exceptions on input limits are handled in specParams module.

11.4.4 Access Routine Semantics

smooth_image(c $\in \mathbb{Z}^{H\times W}$, kernel_sz $\in \mathbb{Z}$, std_deviation $\in \mathbb{R}$):

 # if method == 1, perform Gaussian Blur with OpenCV
img_blur = GaussianBlur(noisy_img, kernel_sz, std_deviation)

- output: img_blur $\in \mathbb{Z}^{h\times w}$
- exception: None

12 MIS of Keypoint Detection Module

[You can reference SRS labels, such as R**??**. —SS] [It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

12.1 Module

detectKeypoints

12.2 Uses

- config (Section 6)
- smoothImage (Section 11)
- imageStruct (Section 15)
- orbStruct (Section 17)

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
load_img	sel_read_path as **str**	img $\in \mathbb{Z}^{h \times w}$	-
detectKeypoints	mthd_kp_detection $\in \mathbb{Z}$, img $\in \mathbb{Z}^{h \times w}$, bin_sz$\in \mathbb{Z}$, patch_sz $\in \mathbb{Z}$, FAST_threshold $\in \mathbb{Z}$	orb_object as **TBD**, keypoints as **key-point tuple**	-

12.4 Semantics

12.4.1 State Variables

- orb_object as **TBD**

12.4.2 Environment Variables

- sel_read_path as **str**

12.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

12.4.4 Access Routine Semantics

load_img(sel_read_path as **str**):
read_path = sel_read_path
    img = cv.imread(sel_read_path)

- output: img $\in \mathbb{Z}^{h \times w}$

- exception: None.

detectKeypoints(mthd_kp_detection $\in \mathbb{Z}$, img $\in \mathbb{Z}^{h \times w}$, bin_sz $\in \mathbb{Z}$, patch_sz $\in \mathbb{Z}$, FAST_threshold $\in \mathbb{Z}$):

- transition: Generate instance of of the detector object

  if mthd_kp_detection == 1

  orb_object = ORB.create(bin_sz $\in \mathbb{Z}$, patch_sz $\in \mathbb{Z}$, FAST_threshold $\in \mathbb{Z}$)

  orb_object.detect(img $\in \mathbb{Z}^{h \times w}$)

- output: Returns orb_object and the set of keypoints $K = \{(x_i, y_i, s_i, \theta_i, r_i) \mid i \in \mathbb{N}\}$, where:
  - $(x_i, y_i) \in \mathbb{R}^2$ (spatial coordinates)
  - $s_i \in \mathbb{R}^+$ (scale)
  - $\theta_i \in [0, 2\pi]$ (orientation)
  - $r_i \in \mathbb{R}$ (response strength)

13 MIS of Feature Descriptor Module

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

13.1 Module

assignDescriptors

13.2 Uses

- detectKeypoints (Section 12)

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
compute_descriptors	mthd_kp_descriptors $\in \mathbb{Z}$ img $\in \mathbb{Z}^{m \times n}$ keypoints as **keypoint tuple**	desc $\in \mathbb{F}_2^{256}$	-

13.4 Semantics

13.4.1 State Variables

- orb_object as **TBD**

13.4.2 Environment Variables

None.

13.4.3 Assumptions

ORB object is instatiated in the Keypoint Detector Module.

13.4.4 Access Routine Semantics

compute_descriptors(orb_obj as **TBD**, img $\in \mathbb{Z}^{h \times w}$, keypoints as keypoint ADT):
desc = orb_object.compute(img, keypoints)

- output: desc $\in \mathbb{F}_2^{256}$
- exception: None

14 MIS of Feature Matching Module

[You can reference SRS labels, such as R**??**. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

14.1 Module

matchFeatures

14.2 Uses

- assignDescriptors (Section 13)

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
create_BF_matcher	mthd_ft_match $\in \mathbb{Z}$	matcher_object as **TBD**	-
match_features	mthd_ft_match $\in \mathbb{Z}$, bf_matcher_object as **(TBD)**, desc1 and desc2 as $\in \mathbb{F}_2^{n\times 256}$	$M = \{m_i \mid m_i = (d_{1i}, d_{2i}, dist_{Hamming})\}$	-
sort_matches	mthd_ft_match $\in \mathbb{Z}$, $M = \{m_i \mid m_i = (d_{1i}, d_{2i}, dist_{Hamming})\}$	matches $\in M = \{m_i \mid m_i = (d_{1i}, d_{2i}, dist_{Hamming})\}$	-

14.4 Semantics

14.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

14.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

14.4.3 Assumptions

Exception handling on user-selected methods and parameters are handled in the Parameter Specification Module.

14.4.4 Access Routine Semantics

create_BF_matcher(mthd_ft_match $\in \mathbb{Z}$):

if mthd_ft_match == 1

- output: bf_matcher_object = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True) # Python syntax
- exception: None

match_features(mthd_ft_match $\in \mathbb{Z}$, bf_matcher_object as **(TBD)**, $desc1$ and $desc2$ as $\in \mathbb{F}_2^{n\times 256}$:
if mthd_ft_match == 1:

- **output:** M = bf_matcher_object.match(desc1, desc2), such that matches $M = \{m_i \mid m_i = (d_{1i}, d_{2i}, dist_{Hamming})\}$, where $d_{1i} \in desc1, d_{2i} \in desc2$, $dist_{Hamming} \in \mathbb{N}$, where $dist_{Hamming}$ is the match distance.
- **exception:** None

sort_matches(mthd_ft_match $\in \mathbb{Z}$, $M = \{m_i \mid m_i = (d_{1i}, d_{2i}, dist_{Hamming})\}$, where $d_{1i} \in desc1, d_{2i} \in desc2$, $dist_{Hamming} \in \mathbb{N}$):
if mthd_ft_match == 1:
matches = bf_matcher_object.sorted(M)

- **output:** matches $\in M = \{m_i | m_i = (d_{1i}, d_{2i}, dist_{Hamming})\}$
- **exception:** None

15 MIS of Image Data Structure Module

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

15.1 Module

imageStruct

15.2 Uses

None.

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
cv.imread	sel_read_path as **str**	img $\in \mathbb{Z}^{h \times w}$	inValidImgPath
cv.imwrite	sel_save_path as **str**, out_img $\in \mathbb{Z}^{h \times w}$	img_png **as .png**	inValidImgPath
cv.drawKeypoints	img_in, overlayImage as $\in \mathbb{Z}^{h \times w}$, keypoints as **keypoint tuple**, colour $\in \mathbb{Z}^3$, flags $\in \mathbb{Z}$	img_kp $\in \mathbb{Z}^{h \times w}$	-
cv.drawMatches	img1_in, img2_in, overlay_image as $\in \mathbb{Z}^{h \times w}$, kp1, kp2 as **keypoint tuple**, matches as **match tuple**, flags $\in \mathbb{Z}$	img_matches $\in \mathbb{Z}^{h \times w}$	-

15.4 Semantics

15.4.1 State Variables

- img_1_name = as **str**
- img_2_name = as **str**

15.4.2 Environment Variables

- read_path as **str**
- save_path as **str**

15.4.3 Assumptions

- ORB objects are initialized prior to use
- BFMatcher objects are initialized prior to use

15.4.4 Access Routine Semantics

cv.imread(sel_read_path as **str**):

- transition: read_path = sel_read_path
- output: img $\in \mathbb{Z}^{h \times w}$
- exception: if no image identified, flag as inValidImgPath

cv.imwrite(sel_save_path as **str**, out_img $\in \mathbb{Z}^{h \times w}$):

- transition: save_path = sel_save_path
- output: img_png **as .png**
- exception: if path is undefined, flag as inValidImgPath

cv.drawKeypoints(img_in as $\in \mathbb{Z}^{h \times w}$, keypoints as **keypoint tuple**, overlayImage $\in \mathbb{Z}^{h \times w}$, colour $\in \mathbb{Z}^3$, flags $\in \mathbb{Z}$):

- output: img_kp $\in \mathbb{Z}^3$

cv.drawMatches(img1_in as $\in \mathbb{Z}^{h \times w}$, kp1 as **keypoint tuple**, img2_in as $\in \mathbb{Z}^{h \times w}$, kp2 as **keypoint tuple**, matches as **match tuple**, overlay_image $\in \mathbb{Z}^{h \times w}$, flags $\in \mathbb{Z}$):

- output: img_matches $\in \mathbb{Z}^{h \times w}$

16 MIS of Image Plot Module

16.1 Module

plotImage

16.2 Uses

- imageStruct (Section 16)

16.3 Syntax

16.3.1 Exported Constants

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
gen_kp_img	img_in, as $\in \mathbb{Z}^{h \times w}$, keypoints as **keypoint tuple**, flags $\in \mathbb{Z}$	img_kp $\in \mathbb{Z}^{h \times w}$	-
gen_matched_features	img_1, img_2 $\in \mathbb{Z}^{h \times w}$, kp_1, kp_2 as **keypoint tuple** matches as **match tuple**, max_matches $\in \mathbb{N}$	img_matches $\in \mathbb{Z}^{h \times w}$	-

16.4 Semantics

16.4.1 State Variables

DrawMatchesFlag $\in \mathbb{Z}$

16.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

16.4.3 Assumptions

- save_kp_img has been initialized with keypoints

16.4.4 Access Routine Semantics

gen_kp_img(img_in, $\in \mathbb{Z}^{h \times w}$, keypoints as **keypoint tuple**):
img_keypoints = cv.drawKeypoints(img_in, keypoints, None, colour=(0, 255, 0), flags=0)

- output: img_keypoints $\in \mathbb{Z}^{h \times w}$

gen_matched_features(img_1, img_2 $\in \mathbb{Z}^{h \times w}$, kp_1, kp_2 as **keypoint tuple**, matches as **match tuple**, max_matches $\in \mathbb{N}$):
img_matches = cv.drawMatches(img_1, kp_1, img_2, kp_2, matches[:max_matches], None, flags=DrawMatchesFlag)

- output: img_matches $\in \mathbb{Z}^{h \times w}$

17 MIS of ORB Data Structure Module

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

17.1 Module

orbStruct

17.2 Uses

None.

17.3 Syntax

17.3.1 Exported Constants

None.

17.3.2 Exported Access Programs

Name	Input	Output	Exceptions
GaussianBlur	img $\in \mathbb{Z}^{h \times w}$, kernel_sz $\in \mathbb{Z}$, std_deviation $\in \mathbb{R}$	smooth_img $\in \mathbb{Z}^{m \times n}$	-
ORB.create	bin_sz $\in \mathbb{Z}$, patch_sz $\in \mathbb{Z}$, FAST_threshold $\in \mathbb{Z}$	orb_object	None
orb_object.detect	img $\in \mathbb{Z}^{h \times w}$	K (set of keypoints)	invalidImg
orb_object.compute	img $\in \mathbb{Z}^{h \times w}$, K where K is a set of keypoints	D (set of descriptors)	invalidImg, invalid-Keypoints

17.4 Semantics

17.4.1 State Variables

orb_object $\in$ **TBD**

17.4.2 Environment Variables

None.

17.4.3 Assumptions

- The input image is a valid grayscale or color image.
- Keypoints are detected before computing descriptors.

17.4.4 Access Routine Semantics

GaussianBlur(img $\in \mathbb{Z}^{h \times w}$, (kernel_sz $\in \mathbb{Z}$, kernel_sz $\in \mathbb{Z}$), std_deviation $\in \mathbb{R}$):

- Output: img_blur $\in \mathbb{Z}^{h \times w}$
- Exception: None. Exceptions are handled in Input Format Module.

ORB.create(bin_sz $\in \mathbb{Z}$, patch_sz $\in \mathbb{Z}$, FAST_threshold $\in \mathbb{Z}$):

- Output: Initializes orb_object as **TBD**
- Exception: None. Exceptions are handled in Input Format Module.

orb_object.detect(img $\in \mathbb{Z}^{h \times w}$):

- Output: Returns the set of keypoints $K = \{(x_i, y_i, s_i, \theta_i, r_i) \mid i \in \mathbb{N}\}$, where:
  - $(x_i, y_i) \in \mathbb{R}^2$ (spatial coordinates)
  - $s_i \in \mathbb{R}^+$ (scale)
  - $\theta_i \in [0, 2\pi]$ (orientation)
  - $r_i \in \mathbb{R}$ (response strength)
- Exception: invalidImage

orb_object.compute(img, K):

- Output: Returns a set of binary descriptors $D = \{d_i \mid d_i \in \mathbb{F}_2^{n \times 256}, i \in \mathbb{N}\}$.
- Exception:
  - image not found $\Rightarrow$ invalidImg
  - keypoints not found $\Rightarrow$ invalidKeypoints

18 MIS of Feature Match Data Module

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

18.1 Module

matchStruct

18.2 Uses

None.

18.3 Syntax

18.3.1 Exported Constants

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
BFMatcher	match_method $\in \mathbb{Z}$, cross_check_flag $\in \mathbb{F}$	brute_force_object as **TBD**	None.
brute_force_object.match	brute_force_object as **TBD**, $D_1, D_2 \in \mathbb{F}_2^{n \times 256}$	Returns a tuple of matches $M = \{m_i \mid m_i = (d_{1i}, d_{2i}, dist_{Hamming})$, such that $d_{1i} \in D_1, d_{2i} \in D_2$, $dist_{Hamming} \in \mathbb{N}$, where $dist_{Hamming}$ is the match distance.	Raises an error if descriptors are invalid or empty.
brute_force_object.sorted	M, as tuple of match objects, key=lambda x: x.distance	Returns a sorted tuple of matches M', where $m'_i = (d_{1i}, d_{2i}, dist_{Hamming})$ matches are sorted in ascending order of $dist_{Hamming}$.	Raises an error if the match set is empty.

18.4 Semantics

18.4.1 State Variables

- brute_force_object as **TBD**

18.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

18.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

- the brute_force_object is initialized before matching is called

18.4.4 Access Routine Semantics

cv.BFMatcher(match_method $\in \mathbb{Z}$, cross_check_flag $\in \mathbb{F}$):

- output: Initializes brute_force_object as **TBD**
- exception: None. Exceptions are handled in Input Format Module.

brute_force_object.match($D_1, D_2 \in \mathbb{F}_2^{n\times 256}$):

- output: Returns matches, $M = \{m_i \mid i \in \mathbb{N}\}$ as a tuple, where each match m_i is defined as $m_i = (d_{1i}, d_{2i}, dist_{Hamming})$ with $d_{1i} \in D_1, d_{2i} \in D_2,\ dist_{Hamming} \in \mathbb{N}$, where $dist_{Hamming}$ represents the match distance.
- exception: Raises an error if the descriptors are invalid or empty.

brute_force_object.sorted(M as **match tuple**, key=lambda x: x.distance):

- output: Returns a sorted tuple of matches M', where matches are sorted in ascending order of distance, $dist_{Hamming}$.
- exception: Raises an error if the match set is empty.

19 MIS of Dataframe Structure Module

[Use labels for cross-referencing —SS]
[You can reference SRS labels, such as R??. —SS]
[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

19.1 Module

dataframeStruct

19.2 Uses

None.

19.3 Syntax

19.3.1 Exported Constants

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

19.4 Semantics

19.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

19.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

19.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

19.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

20 MIS of [Module Name —SS]

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

20.1 Module

[Short name for the module —SS]

20.2 Uses

20.3 Syntax

20.3.1 Exported Constants

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

20.4 Semantics

20.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

20.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

20.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

20.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

20.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

21 Appendix

[Extra information if required —SS]