Implementierung eines einfachen Fahrzeug-Surround-Sicht-Panoramasystems

Zhao Liang/ 2019-10-05

Kategorie: Autonomes Fahren / **Tags**: OpenCV, Python, Rundumsicht-System / **Wörter**: 5719

Link: http://pywonderland.com/surround-view-system-introduction/

Es gibt bereits eine Menge Informationen über das Panoramasichtsystem von Fahrzeugen im Internet, aber es gibt fast keinen Code als Referenz, was für Neuankömmlinge sehr unfreundlich ist. Der Zweck dieses Projekts ist es, das Prinzip des Panoramasystems vorzustellen und eine Python-Implementierung mit vollständigen Basiselementen zu geben, die zu Ihrer Referenz tatsächlich ausgeführt werden kann. Das mit dem Surround-View-System verbundene Wissen ist nicht kompliziert. Die Leser müssen lediglich die Kamerakalibrierung und Perspektiventransformation verstehen und wissen, wie man OpenCV verwendet. Der Projektcode wird auf github gepflegt:

https://github.com/neozhaoliang/surround-view-system-introduction

Dieses Programm wurde ursprünglich auf einem unbemannten Auto entwickelt, das mit einem AGX Xavier ausgestattet war. Die Ergebnisse sind wie folgt:

http://pywonderland.com/images/cn/surroundview/smallcar.mp4

Der Wagen ist mit vier USB-Surround-Fisheye-Kameras ausgestattet. Die Auflösung des von der Kamera zurückgegebenen Bildes beträgt 640x480. Das Bild wird zunächst auf Verzerrungen korrigiert, dann unter projektiver Transformation in eine Vogelperspektive des Bodens umgewandelt und schließlich zusammengefügt und geglättet. Der obige Effekt. Der gesamte Prozess wird in der CPU verarbeitet, und der Gesamtablauf ist reibungslos.

Später habe ich den Code umstrukturiert und auf einen PKW transplantiert (der Prozessor ist das gleiche Modell von AGX), wobei ich fast den gleichen Effekt erhielt:

http://pywonderland.com/images/cn/surroundview/car.mp4

Diese Version verwendet vier 960x640 csi-Kameras, die Ausgabeauflösung des Panoramas beträgt 1200x1600, die Laufgeschwindigkeit des Panoramaverarbeitungs-Threads beträgt etwa 17 fps, wenn die Verarbeitung des Helligkeitsausgleichs nicht durchgeführt wird, und nachdem die Verarbeitung des Helligkeitsausgleichs hinzugefügt wurde, sinkt sie auf nur 7 fps . Ich denke, wenn die Auflösung entsprechend reduziert wird (z.B. kann die Ausgabe mit 480x640 die Anzahl der Pixel auf 1/6 reduzieren), sollten auch glatte visuelle Effekte erzielt werden.

Hinweis: Der schwarze Teil des Bildes ist der blinde Fleck nach der Kameraprojektion. Dies liegt daran, dass die Frontkamera auf der linken Seite der Fahrzeugfront installiert ist und in einem Winkel geneigt ist, um die Fahrzeugmarkierungen zu vermeiden, so dass das Sichtfeld eingeschränkt ist. Stellen Sie sich eine Person vor, die mit geneigtem Hals und schrägen Augen geht...

Die Umsetzung dieses Projekts ist relativ grob. Es dient lediglich als Demonstrationsprojekt, um die Grundelemente der Erzeugung einer Panoramasicht zu zeigen, und jeder kann den Geist verstehen. Der Zweck meiner Entwicklung dieses Projekts besteht darin, die Flugbahn des Fahrzeugs während des automatischen Parkens in Echtzeit darzustellen, und es soll auch als Praktikumsprojekt für die von mir betreuten Praktikanten dienen. Da es keine früheren Erfahrungen und Referenzen gibt, sind die meisten Algorithmen und Prozesse durch Denken geschrieben, nicht unbedingt clever, bitte beraten Sie. Der Code ist in Python geschrieben und ist sicherlich nicht so effizient wie C++, so dass er sich nur zum Lernen und Verifizieren von Ideen eignet.

Lassen Sie mich meine Implementierungsschritte der Reihe nach vorstellen.

Hardware- und Software-Konfiguration

Ich möchte zunächst betonen, dass die Hardwarekonfiguration bei diesem Projekt am wenigsten stören muss. Die im ersten Autoprojekt verwendete Hardware ist wie folgt:

1. Vier USB-Fisheye-Kameras, die unterstützten Auflösungen sind 640x480|800x600|1920x1080. Ich bin hier, weil ich es in Echtzeit unter Python ausführen muss, und die Auflösung ist aus Effizienzgründen auf 640x480 eingestellt.
2. Ein AGX Xavier. Tatsächlich kann ein normaler Laptop ausrutschen und fliegen.

Die für das zweite PKW-Projekt verwendete Hardware ist wie folgt:

1. Vier csi-Kameras, die Auflösung ist auf 960x640 eingestellt.
2. Ein AGX Xavier, das Modell ist dasselbe wie das obige PKW-Projekt, aber ein Industriecomputer wird hinzugefügt, um das csi-Kamerabild zu empfangen.

Ich denke, solange man vier Sichtfelder hat, die ausreichen, um die Kameras um das Auto herum abzudecken, plus ein gewöhnlicher Laptop reicht für die gesamte Offline-Entwicklung.

Die Softwarekonfiguration ist wie folgt:

1. Betriebssystem Ubuntu 16.04/18.04.
2. Python>=3.
3. OpenCV>=3.
4. PyQt5.

Unter ihnen wird PyQt5 hauptsächlich zur Implementierung von Multithreading verwendet, was sich für eine zukünftige Portierung auf die Qt-Umgebung anbietet.

Mehrere vom Projekt verabschiedete Konventionen

Der Einfachheit halber werden in diesem Projekt vier Vermessungskameras jeweils `front`, `back`, `left` und `right` eingesetzt, um sich auf eine Gerätenummer zu beziehen und anzunehmen, dass diese einer ganzen Zahl entspricht, wie z.B. 0, 1, 2, 3. In der aktuellen Entwicklung, bitte modifizieren Sie diese entsprechend der spezifischen Situation.

Die interne Parametermatrix der Kamera wird als Matrix `camera_matrix` a 3x3 bezeichnet. Der Verzerrungskoeffizient wird als `dist_coeffs` bezeichnet, dies ist ein 1x4-Vektor. Bezeichnen die Projektionsmatrix der Kamera `project_matrix`, die eine 3x3-Projektionsmatrix ist.

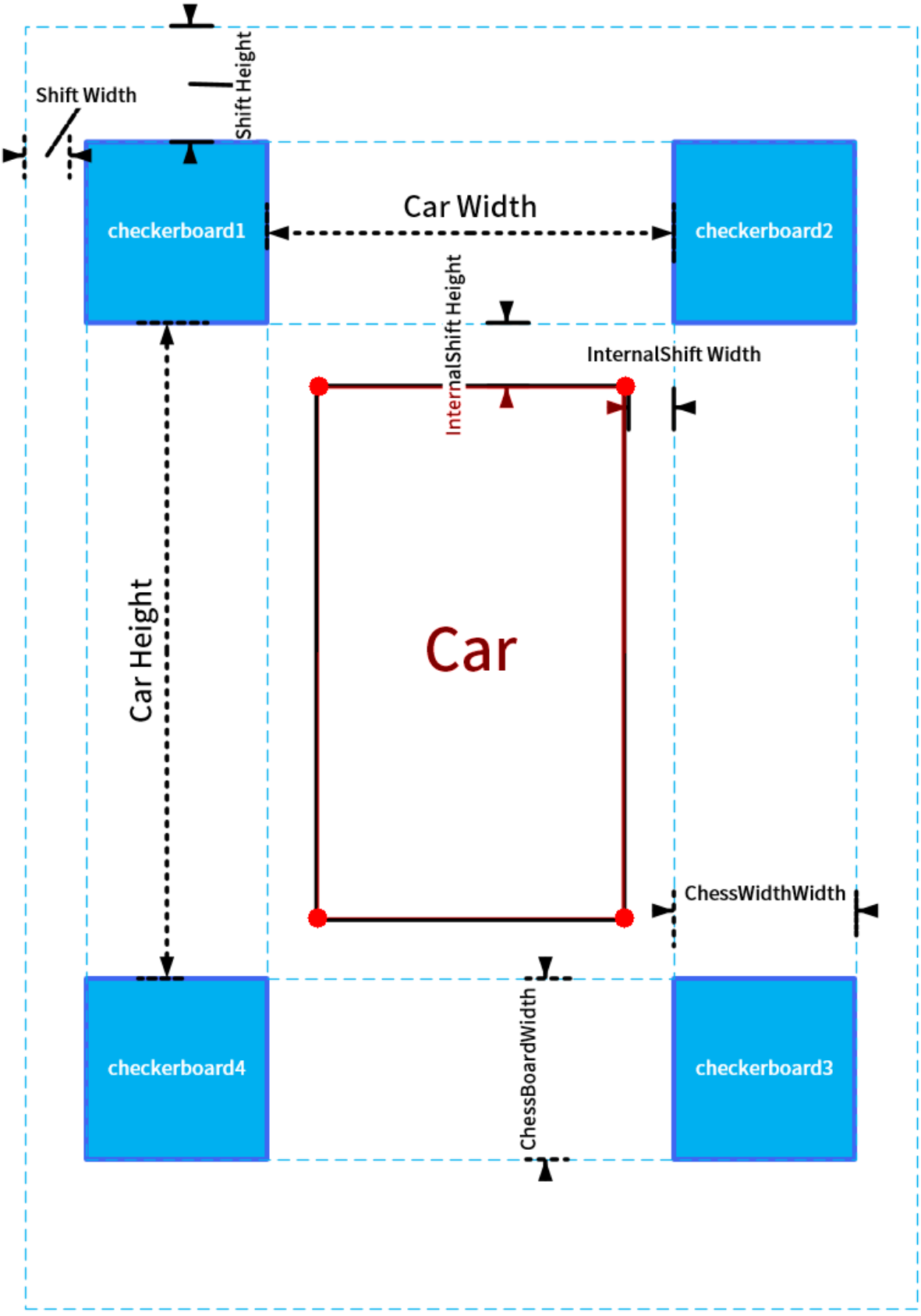
Projektionsbereich und Parameter einstellen

Als nächstes müssen wir die Projektionsmatrix jeder Kamera auf den Boden erhalten. Diese Projektionsmatrix wird das korrigierte Bild der Kamera in eine Vogelperspektive eines rechteckigen

Bereichs auf dem Boden umwandeln. Die Projektionsmatrizen der vier Kameras sind nicht unabhängig voneinander, sie müssen sicherstellen, dass die projizierten Bereiche exakt zusammengesetzt werden können.

Dieser Schritt wird durch eine gemeinsame Kalibrierung erreicht, d.h. durch das Platzieren von Kalibrierobjekten auf dem Boden um das Fahrzeug herum, das Aufnehmen von Bildern, die manuelle Auswahl entsprechender Punkte und die anschließende Erstellung der Projektionsmatrix.

Siehe Abbildung unten:



Legen Sie zunächst vier Kalibrierungstafeln an den vier Ecken der Fahrzeugkarosserie an. Es gibt keine besonderen Anforderungen an die Größe des Musters auf den Kalibriertafeln, solange sie die gleiche Größe haben und auf dem Bild deutlich zu erkennen sind. Jede Kalibriertafel sollte sich genau im Überlappungsbereich der beiden benachbarten Kamerablickfelder befinden.

In dem oben aufgenommenen Kamerabild ist ein Kalibriertuch um den Wagen gespannt. Es ist nicht wichtig, ob es sich um eine Kalibrierplatte oder ein Kalibriertuch handelt, solange die charakteristischen Punkte deutlich zu sehen sind.

Dann müssen wir mehrere Parameter einstellen: (alle folgenden Parameter sind in Zentimetern angegeben)

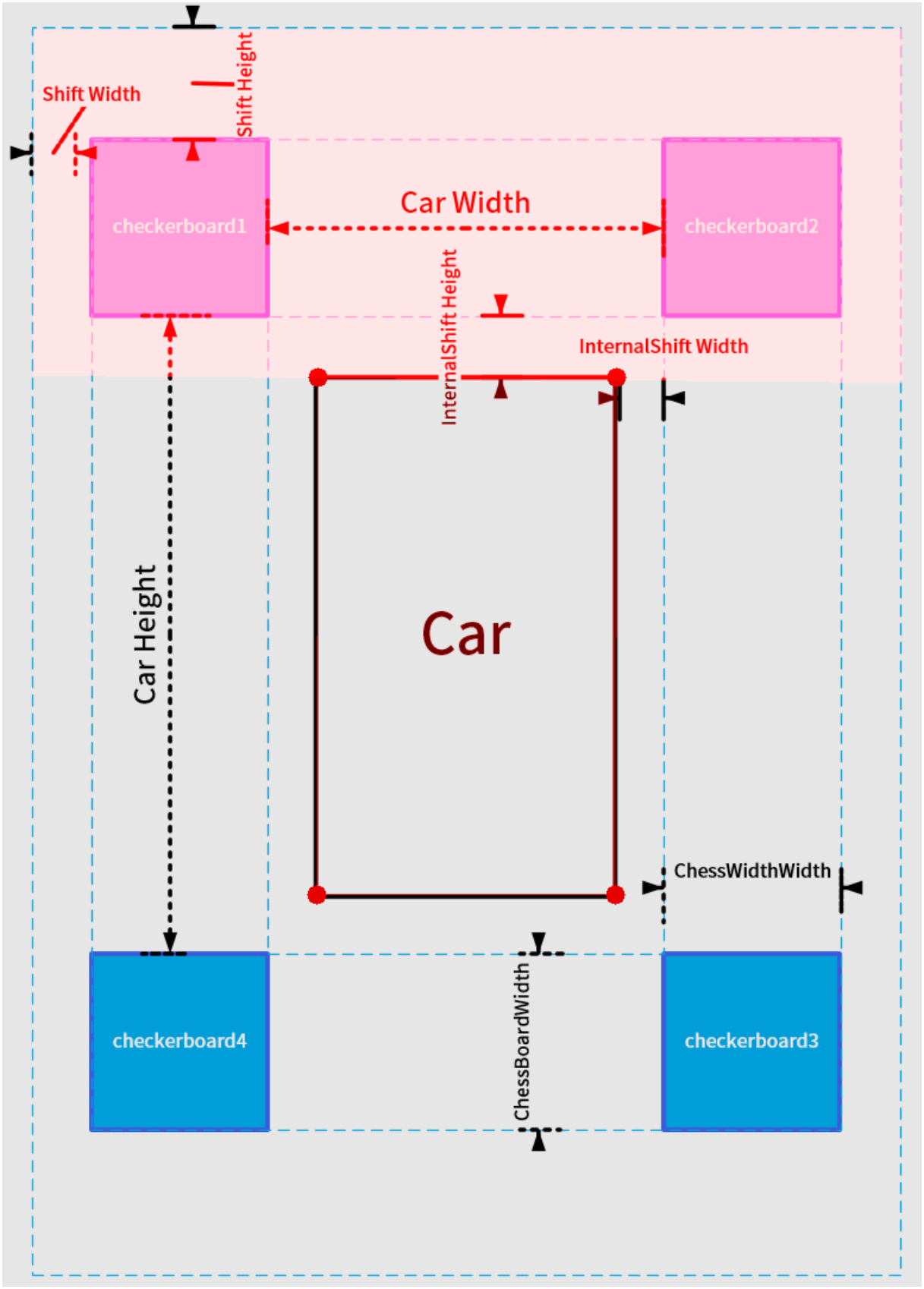
- `innerShiftWidth`, `innerShiftHeight` : The Abstand zwischen der Innenkante der Kalibrierplatte und der linken und rechten Seite des Fahrzeugs sowie der Abstand zwischen der Innenkante der Kalibrierplatte und der Vorder- und Rückseite des Fahrzeugs.
- `shiftWidth`, `shiftHeight`: Diese beiden Parameter bestimmen, wie weit der Blick aus der Vogelperspektive über die Kalibriertafel hinausgehen soll. Je größer diese beiden Werte sind, desto größer ist die Sicht aus der Vogelperspektive zu sehen. Dementsprechend ist die Verformung der entfernten Objekte nach der Projektion gravierender, so dass Sie entsprechend Ihren Umständen wählen sollten.
- `totalWidth`, `totalHeight`: Diese beiden Parameter stellen die Gesamtbreite und -höhe der Vogelperspektive dar. In unserem Projekt beträgt die kalibrierte Tuchbreite 6 m und die Höhe 10 m, so dass die Reichweite des Bodens in der Vogelperspektive `(600 + 2 * shiftWidth, 1000 + 2 * shiftHeight)` beträgt. Um die Berechnung zu vereinfachen, lassen wir jedes Pixel 1 cm entsprechen, so dass die Gesamtbreite und -höhe der Vogelperspektive
  `totalWidth = 600 + 2 * shiftWidth`
  `totalHeight = 1000 + 2 * shiftHeight`
- Das Fahrzeug befindet sich in den vier Ecken des rechteckigen Bereichs (mit roten Punkten markiert), die vier Eckkoordinaten des Punktes sind `(xl, yt)`, `(xr, yt)`, `(xl, yb)`, `(xr, yb)` (`l` drückt left aus, `r` zeigt right an, `t` repräsentiert ein top, `b` zeigt bottom). Dieser rechteckige Bereich ist für die Kamera unsichtbar, und wir werden ihn mit einem Fahrzeugsymbol abdecken.

Beachten Sie, dass die Verlängerungslinien auf den vier Seiten dieses Fahrzeugbereichs die gesamte Ansicht aus der Vogelperspektive in acht Teile aufteilen, von denen FL (Bereich I), FR (Bereich II), BL (Bereich III), BR (Bereich IV) die sich überschneidenden Bereiche des Sichtfeldes benachbarter Kameras sind, die auch unsere Hauptanforderungen sind Der Teil, in dem der Fusionsprozess durchgeführt wird. Die vier Bereiche F, R, L und R gehören zum separaten Sichtfeld jeder Kamera und müssen nicht verschmolzen werden.

Die oben genannten Parameter sind in param_settings.py gespeichert:

https://github.com/neozhaoliang/surround-view-system-introduction/blob/master/surround_view/param_settings.py

Nach der Einstellung der Parameter wird der Projektionsbereich jeder Kamera bestimmt. Die Projektionsfläche, die der Frontkamera entspricht, ist zum Beispiel wie folgt:



Als nächstes müssen wir die Orientierungspunkte manuell auswählen, um die Projektionsmatrix auf den Boden zu erhalten.

Manuelle Kalibrierung zum Erhalt der Projektionsmatrix

Führen Sie zunächst das Skript run_get_projection_maps.py im Projekt aus:

https://github.com/neozhaoliang/surround-view-system-introduction/blob/master/run_get_projection_maps.py

Für dieses Skript müssen Sie die folgenden Parameter eingeben:

- `-camera`: Geben Sie an, um welche Kamera es sich handelt.
- `-scale`: Das horizontale und vertikale Zoom-Verhältnis des Bildschirms nach der Korrektur.
- `-shift`: Die horizontale und vertikale Verschiebungsdistanz der Bildschirmmitte nach der Korrektur.

Warum brauchen wir diese beiden Parameter `scale` und `shift`? Das liegt daran, dass die Standardmethode der OpenCV-Korrektur darin besteht, einen Bereich im Bild, den OpenCV in dem von der Fischaugen-Kamera korrigierten Bild als angemessen "erachtet", zuzuschneiden und zurückzugeben. Dadurch gehen unweigerlich einige Pixel verloren, insbesondere derjenige, den wir auswählen wollen. Feature-Punkte werden abgeschnitten. Glücklicherweise erlaubt es uns die Funktion cv2.fisheye.initUndistortRectifyMap (https://docs.opencv.org/master/db/d58/group__calib3d__fisheye.html#ga0d37b45f780b32f63ed19c21aa9fd333), eine neue interne Parametermatrix einzugeben, um das Bild nach der Korrektur, aber vor dem Beschneiden zu vergrößern und zu verkleinern. Sie können versuchen, das geeignete horizontale und vertikale Kompressionsverhältnis und die Position der Bildmitte so einzustellen und zu wählen, dass die Landmarke auf dem Boden in einer komfortablen Position auf dem Bildschirm erscheint, um die Kalibrierung zu erleichtern.

Führen Sie Folgendes aus:

```
python run_get_projection_maps.py -camera front -scale 0.7 0.8 -shift -150 -100
```

Der korrigierte Bildschirm der Frontkamera sieht wie folgt aus:

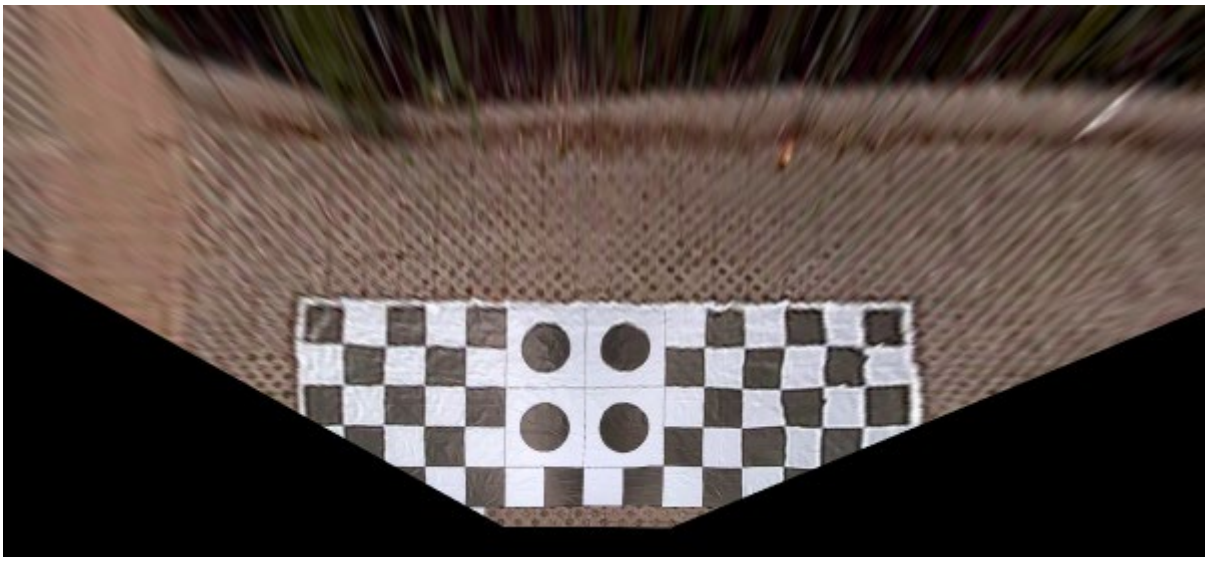
Dann klicken Sie die vier vorgegebenen Markierungspunkte der Reihe nach an (die Reihenfolge kann nicht falsch sein!), das Ergebnis ist wie folgt:

Beachten Sie, dass der Einstellcode der Markierung hier steht:

https://github.com/neozhaoliang/surround-view-system-introduction/blob/master/surround_view/param_settings.py#L40

Diese vier Punkte können frei gesetzt werden, aber Sie müssen ihre Pixelkoordinaten in der Vogelperspektive im Programm manuell ändern. Wenn Sie auf diese vier Punkte in der Kalibrierungskarte klicken, berechnet OpenCV eine Projektionsmatrix auf der Grundlage der Übereinstimmung zwischen ihren Pixelkoordinaten in der Kalibrierungskarte und den Pixelkoordinaten in der Vogelperspektive. Das hier verwendete Prinzip besteht darin, eine projektive Transformation zu bestimmen, die vier Punkten entspricht (vier Punkte, die acht Gleichungen entsprechen, können angegeben werden, um die acht Unbekannten der projektiven Matrix zu lösen. Beachten Sie, dass die letzte Komponente der projektiven Matrix immer auf 1 fixiert ist).

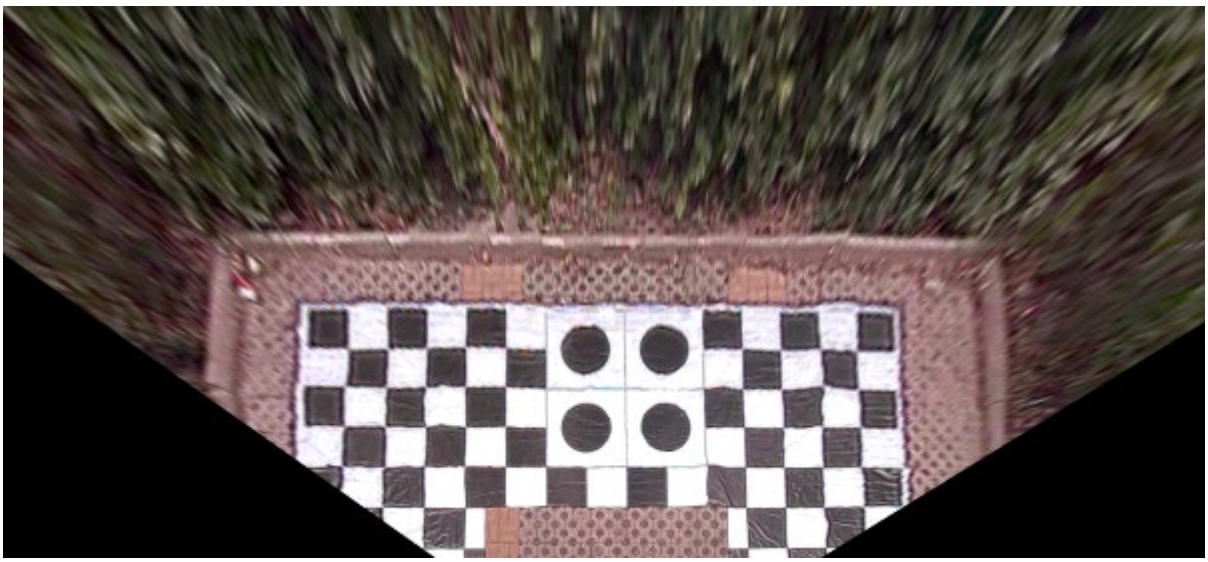
Wenn Sie nicht aufpassen, dass Sie schief liegen, dann können Sie dthe drücken, um einen falschen Punkt zu löschen. Nachdem Sie ihn ausgewählt haben, klicken Sie auf Enter, und das projektive Rendering wird angezeigt:



Damit das Ergebnis dann die Eingabetaste drücken kann, schreibt es die Projektionsmatrix `front.yaml`, die Matrix ist der Name `project_matrix`. Es gelingt ihm nicht, erneut `q` exit zu drücken.

Ein weiteres Beispiel ist die Kalibrierung der Rückfahrkamera, wie in der Abbildung unten gezeigt:

Das entsprechende Projektionsbild ist

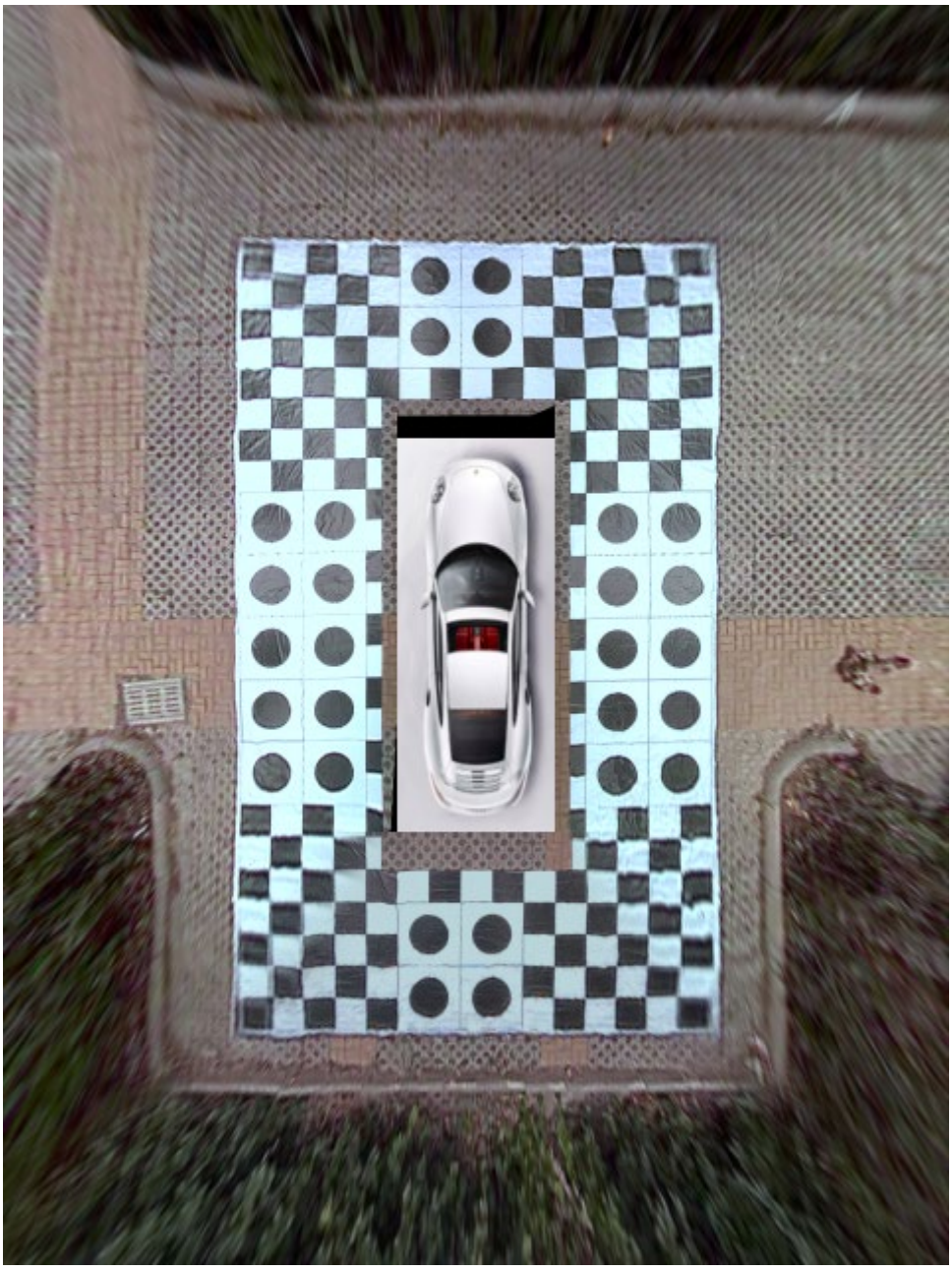


Mit dieser Operation für die vier Kameras erhalten wir die Vogelperspektive auf die vier Kameras und die entsprechenden vier Projektionsmatrizen. Unsere nächste Aufgabe besteht darin, diese vier Luftaufnahmen zusammenzufügen.

Stitchen und Glätten der Vogelperspektive

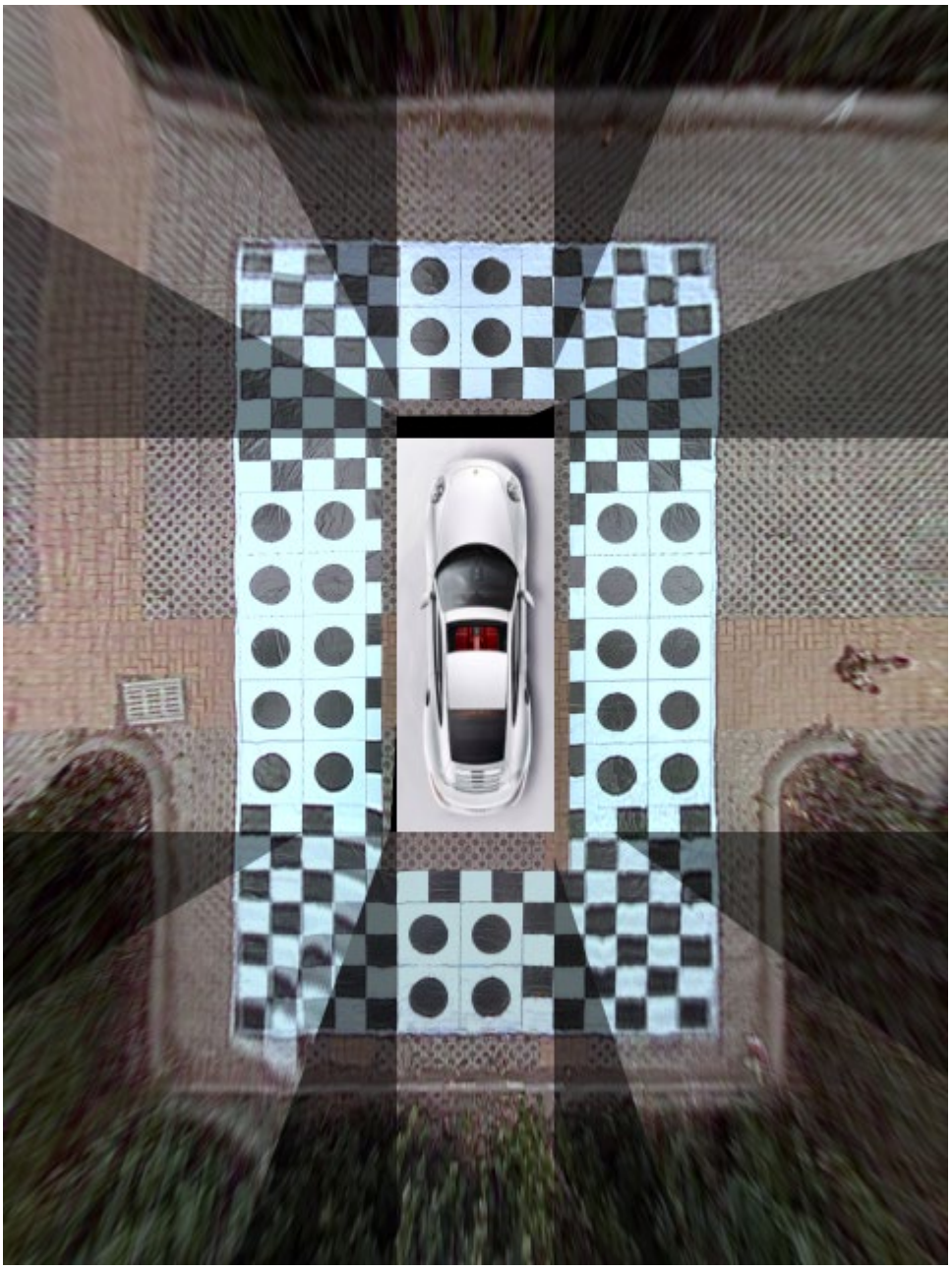
Wenn alles für Ihre vorherigen Operationen normal ist, sollte das folgende Mosaik angezeigt werden, nachdem run_get_weight_matrices.py ausgeführt wurde:

https://github.com/neozhaoliang/surround-view-system-introduction/blob/master/run_get_weight_matrices.py



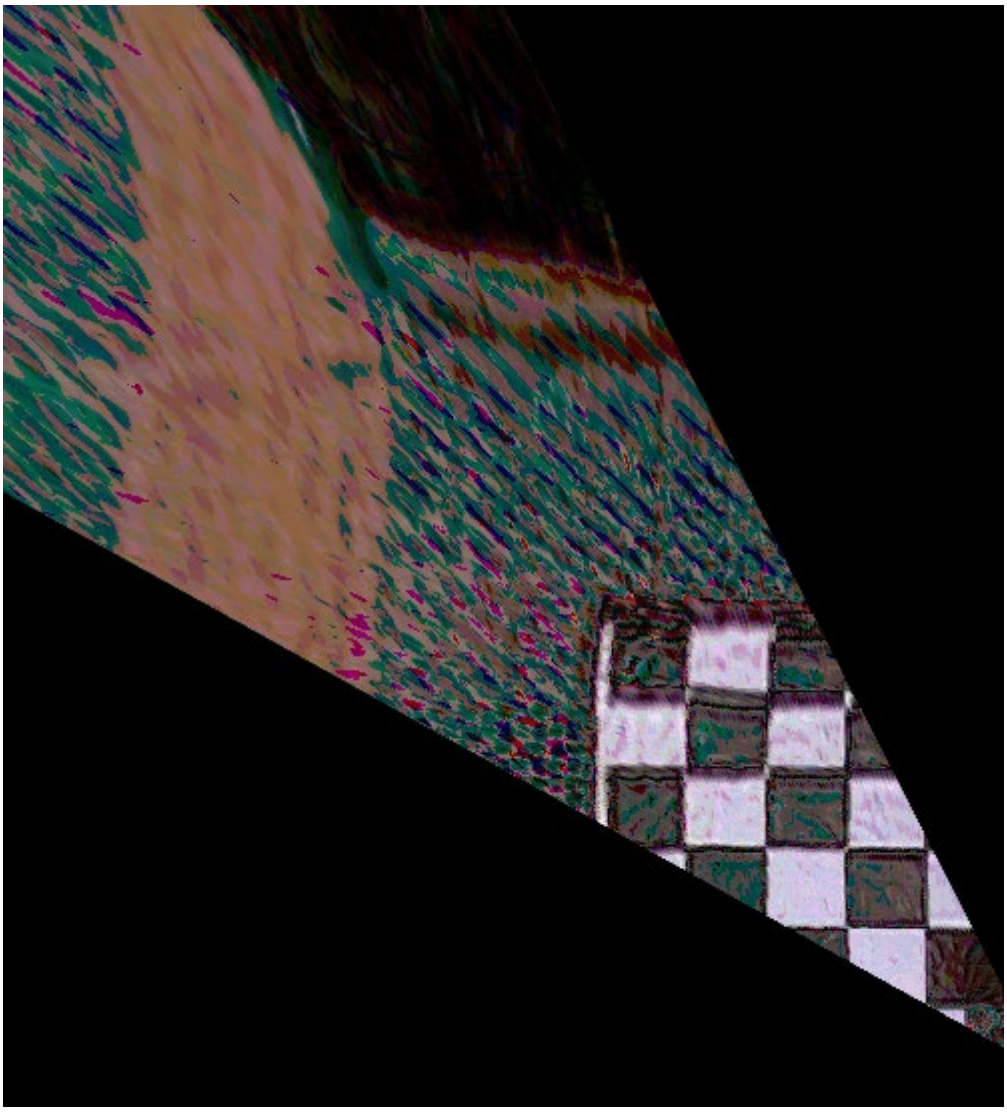
Lassen Sie mich Schritt für Schritt einführen, wie das geschieht:

1. Da es Überlappungsbereiche zwischen benachbarten Kameras gibt, ist die Verschmelzung dieses Teils der Schlüssel. Wenn Sie direkt den gewichteten Mittelwert der beiden Bilder bilden (die Gewichtung beträgt jeweils 1/2), erhalten Sie ein Ergebnis ähnlich dem folgenden:

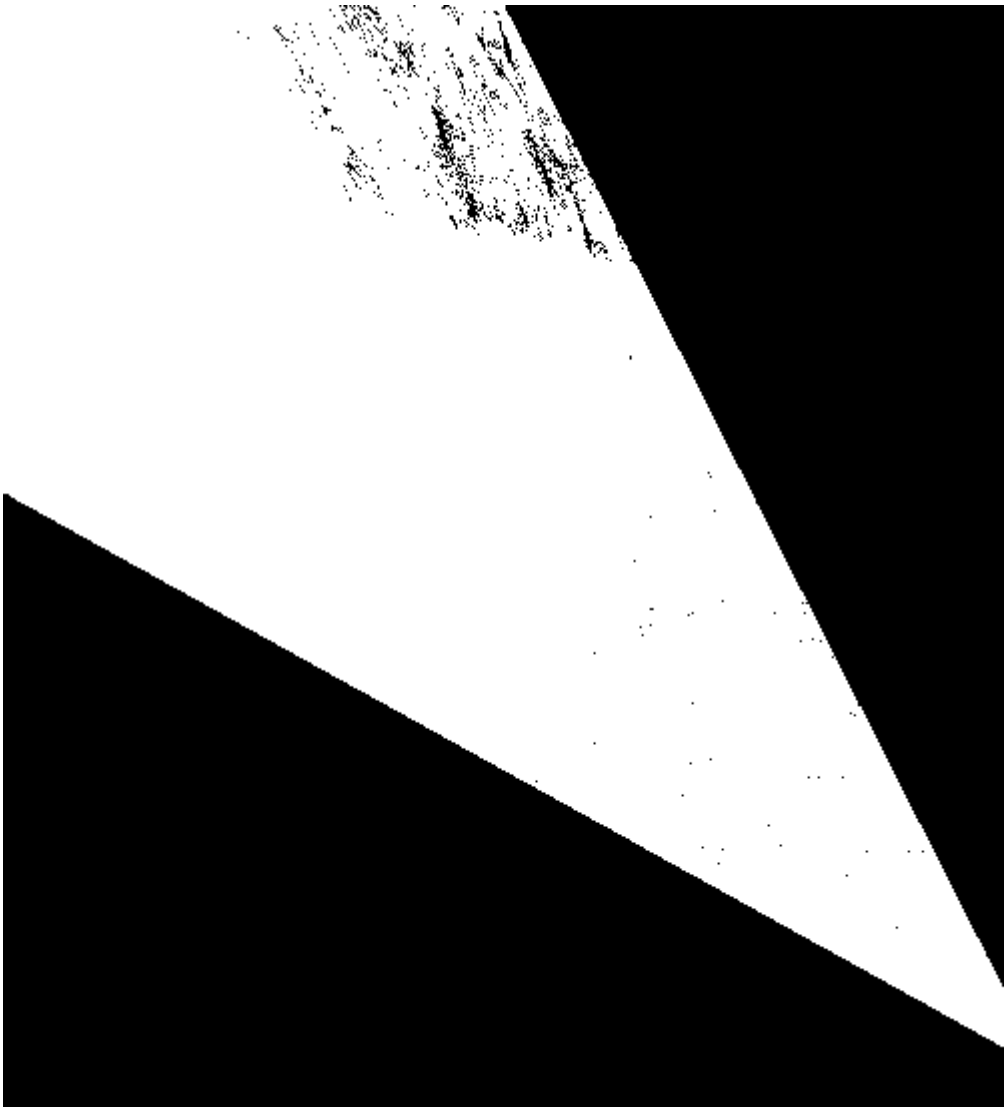


Sie können sehen, dass aufgrund des Korrektur- und Projektionsfehlers die Projektionsergebnisse benachbarter Kameras im deckungsgleichen Bereich nicht vollständig konsistent sind, was zu Verstümmelungen und Geisterbildern im Stitching-Ergebnis führt. Der Schlüssel liegt hier darin, dass sich der Gewichtskoeffizient mit der Änderung des Pixels ändern sollte und sich kontinuierlich mit dem Pixel ändert.

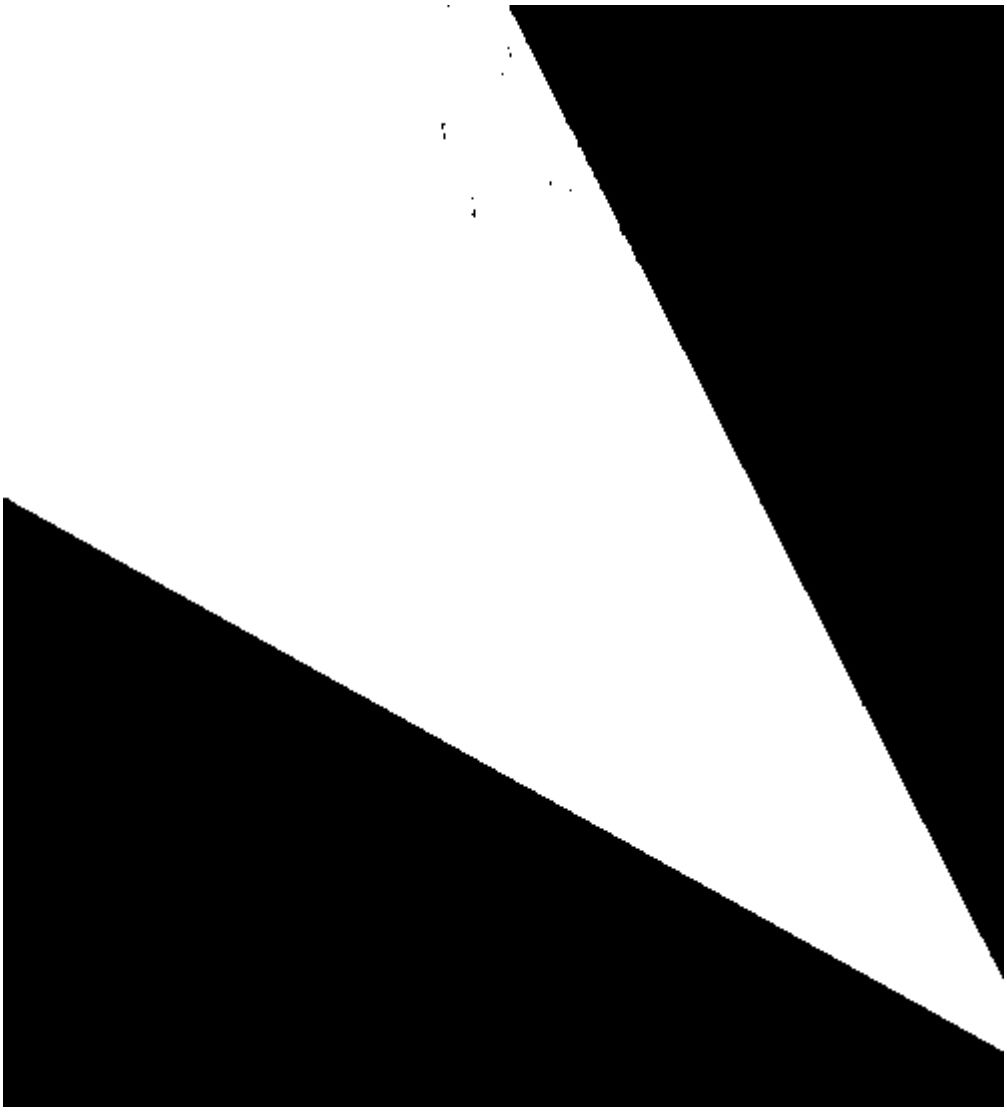
2. In der linken oberen Ecke des Bereichs ist dieser Bereich beispielsweise vorne, links der überlappende Bereich des Sichtfelds der beiden Kameras. Wir nehmen zuerst den überlappenden Teil des Projektionsbildes heraus:



Vergrauung und Binarisierung:

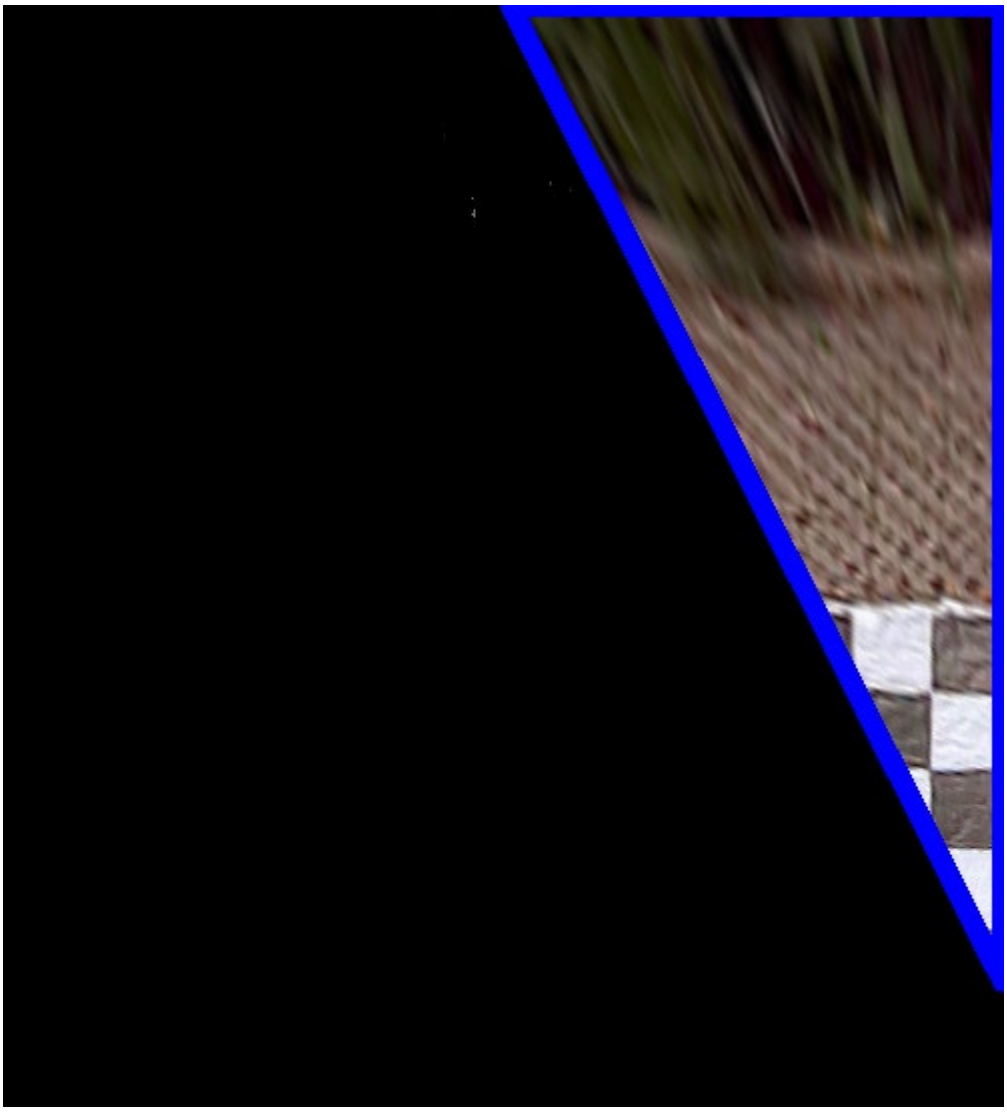


Beachten Sie, dass es Lärm enthält, der durch morphologische Operationen entfernt werden kann (er muss nicht besonders detailliert sein, er kann grob entfernt werden):

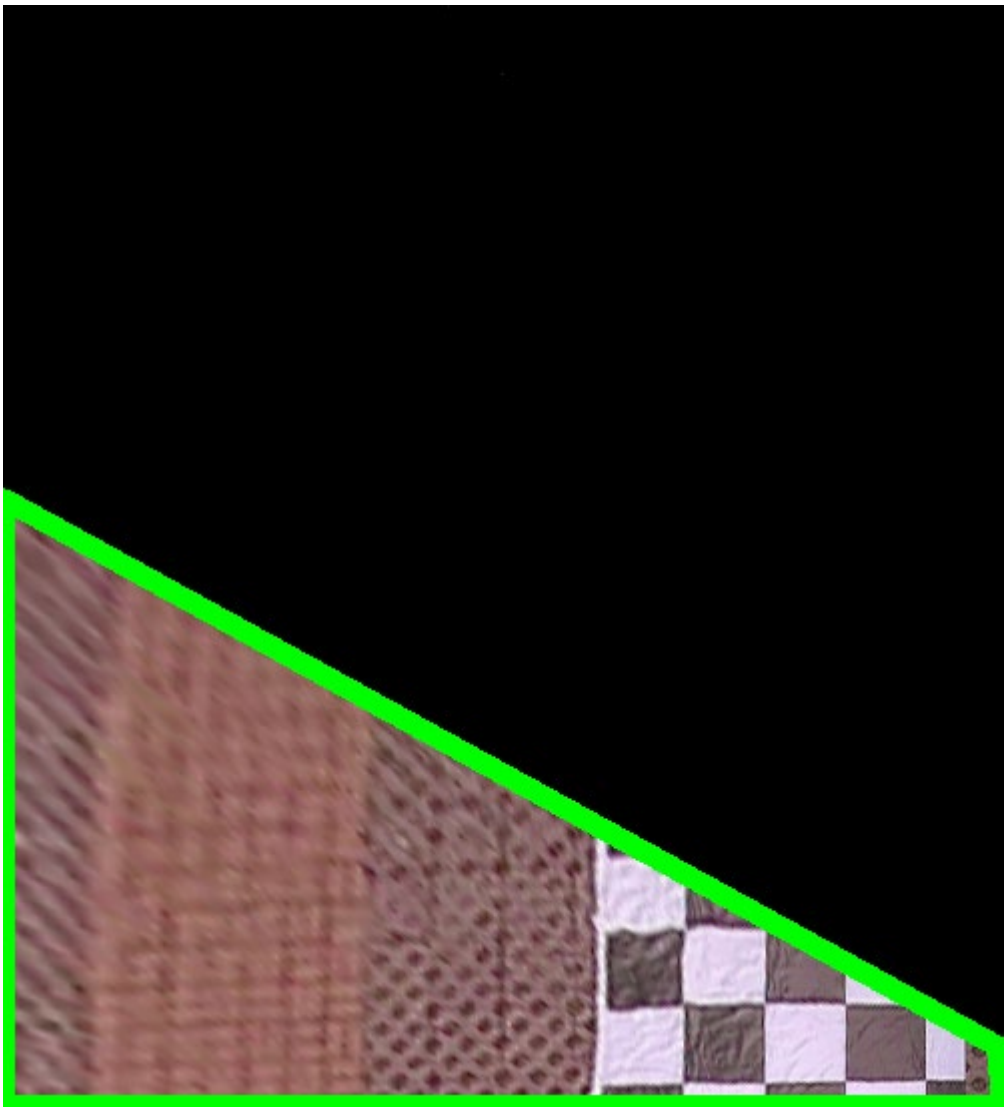


Bislang haben wir eine vollständige Maske des Überlappungsbereichs erhalten.

3. Die `front`, `left` des Bilds von jedem befindet sich außerhalb der Überlappung Region Grenze erkannt, ist dieser Schritt zunächst durch `cv2.findContours` Erhalt der äußersten Grenze, dann die `cv2.approxPolyDP` erhalten polygonalen Kontur Approximation aufgerufen:



Vorher



Nachher

Wir platzieren die `front` Kamera nach Subtraktion des Profils der mit `polyA` bezeichneten Überlappungsregion (linkes Bild über blauem Rand), das `left` Kameraprofil nach Subtraktion der mit `polyB` bezeichneten Überlappungsregion (rechtes Bild über grünem Rand).

4. Für jedes Pixel in der Überlappungsregion unter Verwendung des `cv2.pointPolygonTest`, der auf die beiden Polygone `polyA` und `polyB` Abstand d_A, d_B berechnet wurde, ist das diesem Pixel entsprechende Gewicht $w = d_B^2/(d_A^2 + d_B^2)$, d.h. wenn das innerhalb `front` Pixel des Bildes ist, ist es gerade weiter von `polyB` entfernt, also der Wert von größeren Rechten.
5. von Pixeln innerhalb des Überlappungsbereichs nicht, wenn sie zum `front` Bereich der Kamera gehören, dann ist ihr Wert 1, andernfalls ist die Gewichtung gleich Null. Wir erhalten also eine sich ständig ändernde Matrix mit Werten von 0 bis 1. G, Das Graustufenbild sieht wie folgt aus:

wird G das verschmolzene Bildl kann als das Gewicht `front * G + (1- G) * left` erhalten werden.

6. Beachten Sie, dass, da der Pixelwert im überlappenden Bereich der gewichtete Mittelwert der beiden Bilder ist, die in diesem Bereich erscheinenden Objekte unweigerlich Geistererscheinungen zeigen. Daher müssen wir den Bereich des überlappenden Bereichs so weit wie möglich komprimieren, und es werden nur die Pixelgewichte für die Naht berechnet, der Kampf um die obere Naht, um die Pixel von `front`, dem ursprünglichen Pixel, zu nutzen, der Kampf um die Pixel, um die untere Naht von `back`, den ursprünglichen Pixeln, zu nutzen. Dieser Schritt kann durch d_B gesteuert werden. Der Wert wird erhalten.
7. Wir haben auch einen wichtigen Schritt verpasst: Aufgrund der unterschiedlichen Belichtung der verschiedenen Kameras wird es in verschiedenen Bereichen einen Helligkeitsunterschied geben, der sich auf die Schönheit auswirkt. Wir müssen die Helligkeit der einzelnen Bereiche so einstellen, dass die Helligkeit des gesamten Mosaikbildes tendenziell konsistent ist. Dieser Schritt ist nicht einzigartig, und es gibt viel Raum für freies Spiel. Ich habe die im Internet erwähnten Methoden geprüft und festgestellt, dass sie entweder zu kompliziert und fast unmöglich in Echtzeit zu realisieren sind; oder zu einfach, um den gewünschten Effekt zu erzielen. Besonders im Beispiel des zweiten Videos oben, weil das Sichtfeld der Frontkamera durch das Autologo blockiert wird, ist der Empfindlichkeitsbereich unzureichend, was zu einem großen Helligkeitsunterschied zu den anderen drei Kameras führt, und es ist sehr schwierig, ihn einzustellen.

Die Grundidee ist folgende: Jede der Kameras lieferte Bilder mit BGR drei Kanäle, vier Kamerabilder lieferten insgesamt 12 Kanäle. Wir müssen 12 Koeffizienten berechnen, diese 12 Koeffizienten mit diesen 12 Kanälen multiplizieren und sie dann zu einem justierten Bild kombinieren. Der zu helle Kanal sollte abgedunkelt werden, so dass der Multiplikationsfaktor kleiner als 1 ist, und der zu dunkle Kanal sollte aufgehellt werden, so dass der Multiplikationsfaktor größer als 1 ist. Diese Koeffizienten lassen sich aus den Helligkeitsverhältnissen der vier Bildschirme in den vier überlappenden Bereichen ableiten. Sie können die Methode zur Berechnung der Koeffizienten frei gestalten, solange dieses Grundprinzip erfüllt ist.
Siehe meine Implementierung hier:
https://github.com/neozhaoliang/surround-view-system-introduction/blob/master/surround_view/birdview.py#L210
Sie fühlt sich an wie ein Stück Shader-Code.
Eine andere Möglichkeit, faul zu sein, besteht darin, eine Tonemapping-Funktion im Voraus zu berechnen (wie eine stückweise lineare oder AES-Tonemapping-Funktion) und dann die Umwandlung aller Pixel zu erzwingen. Diese Methode ist die arbeitssparendste, aber der resultierende Bildton wird sich von der realen Szene unterscheiden. Es scheint, dass einige Produkte auf dem Markt diese Methode verwenden.

8. Da die Intensität der verschiedenen Kanäle der Kamera in einigen Fällen unterschiedlich ist, ist es schließlich notwendig, eine Farbbalance durchzuführen, wie in der Abbildung unten dargestellt:

Weiterverarbeitung originaler Stitching-Screen

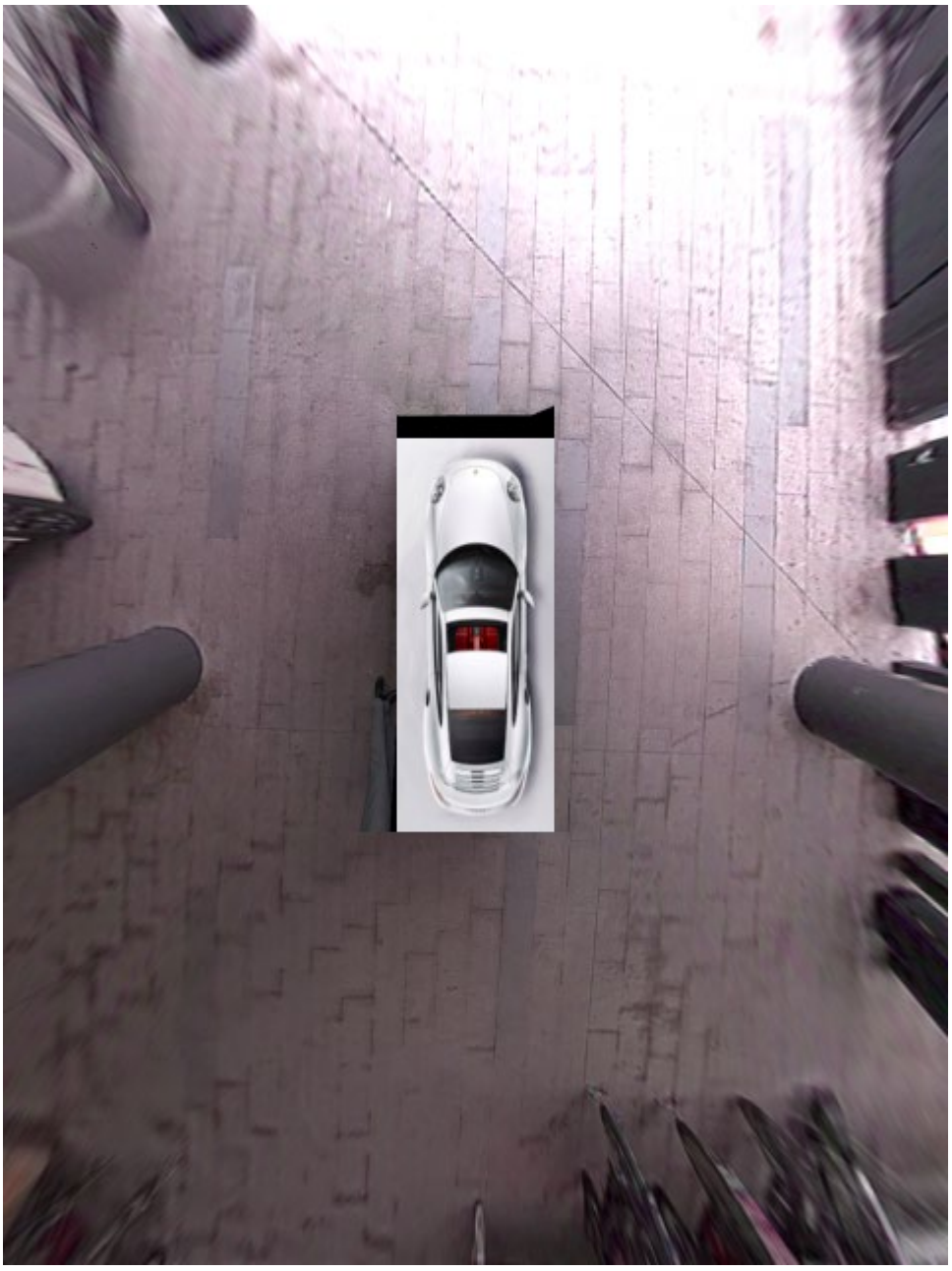


Bild nach Helligkeitsabgleich

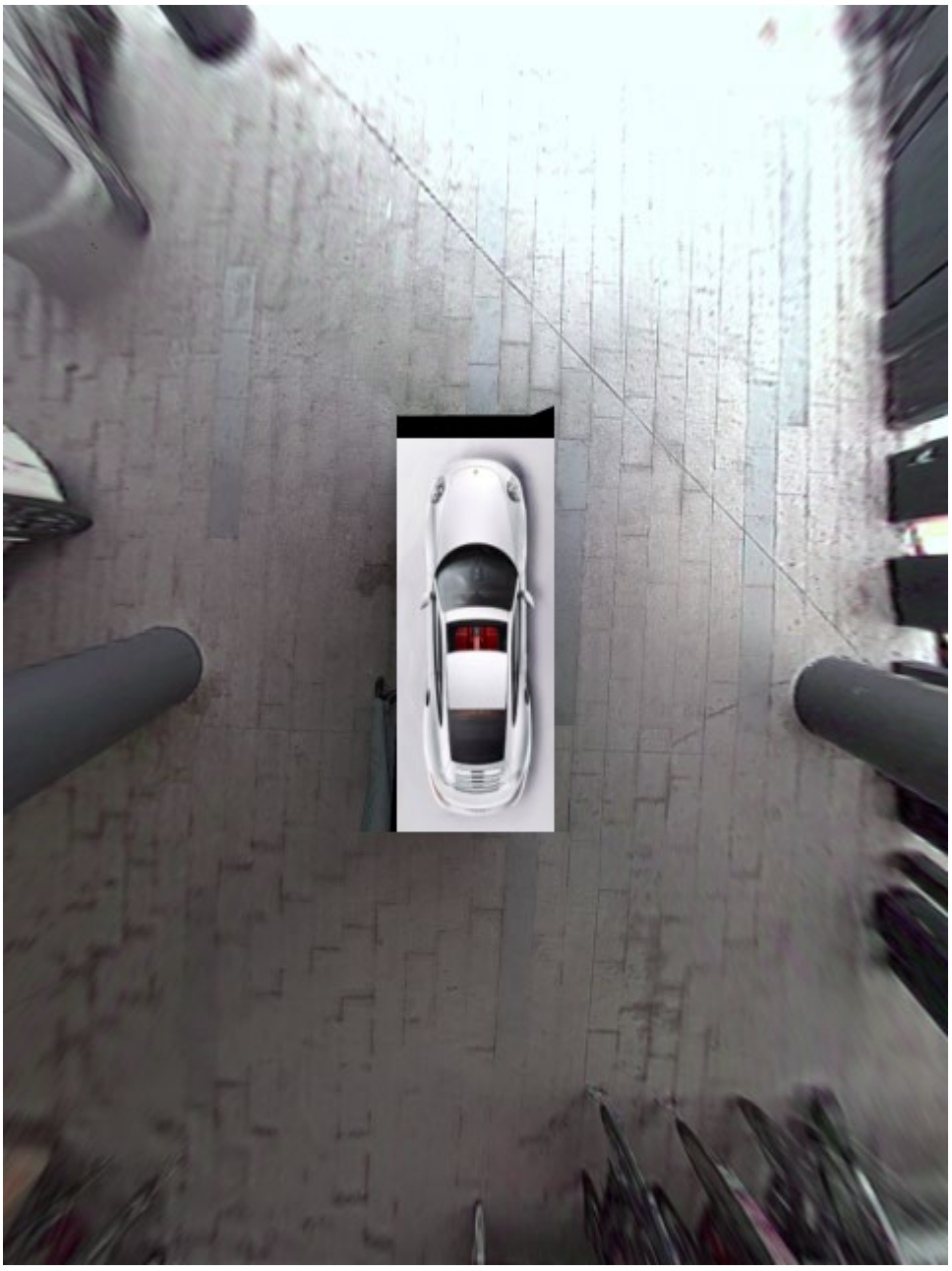


Bild nach dem Farbausgleich

Im zweiten Videobeispiel ist die Farbe des Bildes rötlich, und das Bild kehrt nach Hinzufügen der Farbbalance zur Normalität zurück.

Überlegungen zur spezifischen Implementierung

1. Multithreading und Thread-Synchronisierung. In den beiden Beispielen in diesem Artikel werden die vier Kameras nicht durch Hardware ausgelöst, um die Synchronisation zu gewährleisten, und selbst wenn die Hardware synchronisiert ist, sind die Verarbeitungsthreads der vier Bildschirme möglicherweise nicht synchronisiert, so dass ein Thread-Synchronisationsmechanismus erforderlich ist. Bei der Implementierung dieses Projekts wird ein primitiverer verwendet, und sein Kerncode lautet wie folgt:

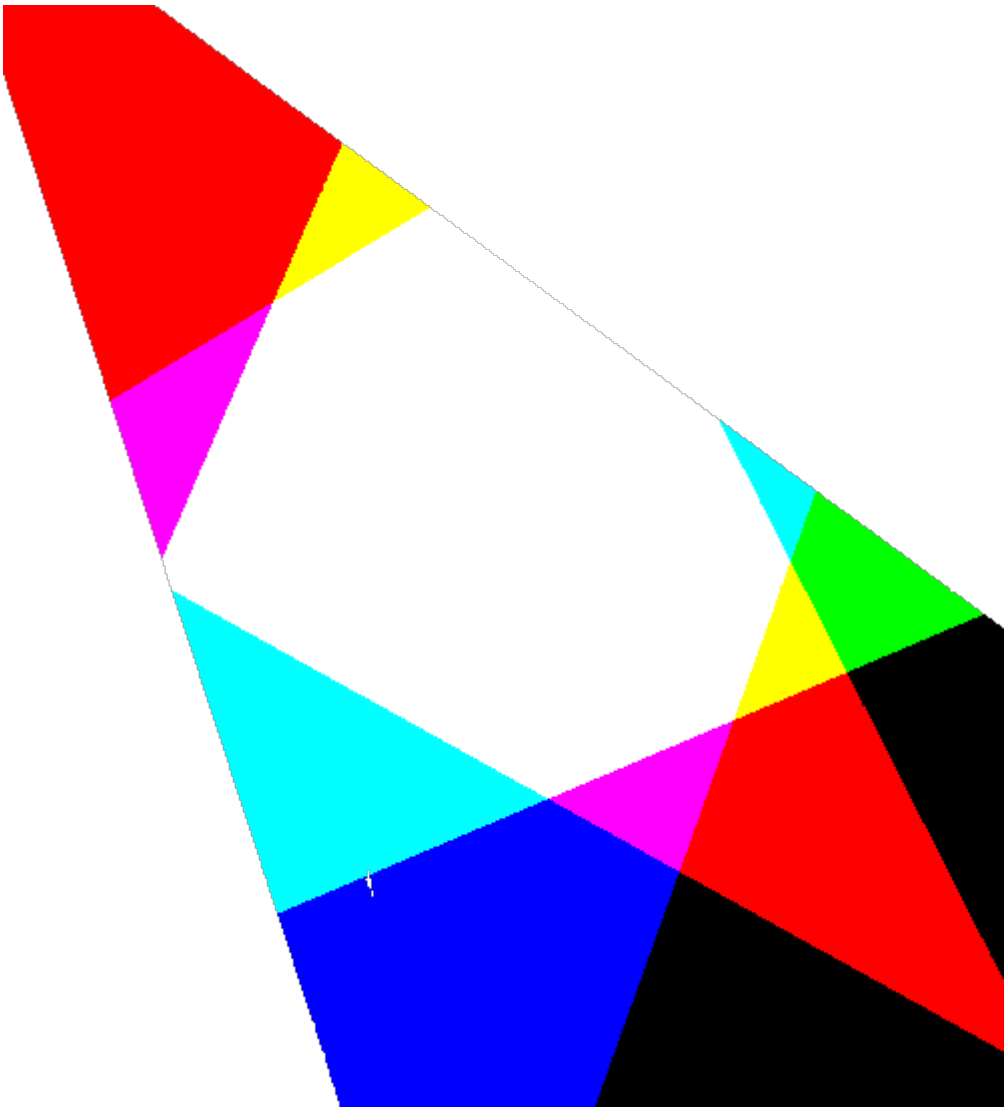
```
class MultiBufferManager(object):

...

def sync(self, device_id):
```

```
    # only perform sync if enabled for specified device/stream
    self.mutex.lock()
    if device_id in self.sync_devices:
        # increment arrived count
        self.arrived += 1
        # we are the last to arrive: wake all waiting threads
        if self.do_sync and self.arrived == len(self.sync_devices):
            self.wc.wakeAll()
        # still waiting for other streams to arrive: wait
        else:
            self.wc.wait(self.mutex)
        # decrement arrived count
        self.arrived -= 1
    self.mutex.unlock()
```

Hier ein `MultiBufferManager` -Ziel, um alle Threads zu verwalten, jede Kamera in jeder Schleife ruft ihre `sync`-Methoden auf, und um das Objekt zu benachrichtigen: "Der Berichtzähler wird durch die Methode inkrementiert, ich habe die letzte Aufgabe abgeschlossen Bitte fügen Sie mich dem Schlaf-Pool hinzu und warten Sie auf die nächste Aufgabe". Sobald der Zähler 4 erreicht, löst er aus, alle Threads aufzuwecken, um in die nächste Runde der Aufgabenschleife einzutreten.

2. Erstellen Sie eine Nachschlagetabelle, um die Berechnung zu beschleunigen. Fischaugen-Objektivbilder müssen korrigiert, projiziert und gespiegelt werden, bevor sie für das Stitchen verwendet werden können. Diese drei Schritte erfordern eine häufige Zuweisung und Zerstörung des Bildspeichers, was sehr zeitaufwendig ist. In meinem Test hat sich der Crawling-Thread immer bei etwas mehr als 30 fps stabilisiert, aber der Verarbeitungs-Thread für jeden Bildschirm beträgt nur etwa 20 fps. Dieser Schritt lässt sich am besten durch die Vorberechnung einer Nachschlagetabelle beschleunigen. Erinnern Sie sich an `cv2.fisheye.initUndistortRectifyMap`? Sie gibt `mapx, mapy` zurück, welche zwei Lookup-Tabellen sind. Wenn Sie beispielsweise den Typ der Matrix angeben, wie sie `cv2.CV_16SC2` zurückgibt, ist `mapx` eine pixelweise Nachschlagetabelle, `mapy` ist eine glatte Interpolation für ein eindimensionales Array (nicht wegzuwerfen). Aus dem gleichen Grund `project_matrix` ist es nicht schwierig, eine Nachschlagetabelle zu erhalten, zwei zusammen können direkt vom Originalbild zu einer Nachschlagetabelle der Projektionswand erhalten werden (natürlich ein Verlust an Informationen, die für die Interpolation verwendet werden). Als Ergebnis dieses Projekts ist in Python implementiert, aber Python's `for` Zyklus-Effizienz ist nicht hoch, so gibt es keine Verwendung einer solchen Lookup-Tabelle.
3. Vier Gewichtungsmatrizen können als `RGBA`-komprimierte vier Kanäle zu einem Bild verwendet werden, so dass sie leicht gespeichert und gelesen werden können. Dasselbe gilt für die Maskenmatrix, die den vier überlappenden Bereichen entspricht:

Gewichtsmatrix



Masken-Matrix

Betrieb des realen Fahrzeugs

Sie können run_live_demo.py auf dem echten Auto ausführen, um den endgültigen Effekt zu überprüfen:

https://github.com/neozhaoliang/surround-view-system-introduction/blob/master/run_live_demo.py

Sie müssen darauf achten, dass Sie die Gerätenummer der Kamera ändern und die Art und Weise, wie OpenCV die Kamera öffnet. usb-Kamera kann direkt verwendet werden cv2.VideoCapture(i) (i eine usb-Gerätenummer) geöffnet wird, ist es notwendig, die Kamera CSI gstreamer geöffnet, den entsprechenden Codes hier:

https://github.com/neozhaoliang/surround-view-system-introduction/blob/master/surround_view/utils.py#L5

https://github.com/neozhaoliang/surround-view-system-introduction/blob/master/surround_view/capture_thread.py#L75

Anhang: Liste der Projektskripte

Die aktuellen Skripte im Projekt sind nach der Reihenfolge ihrer Ausführung wie folgt geordnet:

1. `run_calibrate_camera.py`: Wird für die kamerainterne Parameterkalibrierung verwendet.
2. `param_settings.py`: Dient zur Einstellung der Parameter der Projektionsfläche.
3. `run_get_projection_maps.py`: Dient zur Einstellung der Parameter der Projektionsfläche: Wird verwendet, um die Projektionsmatrix manuell auf den Boden zu kalibrieren.
4. `run_get_weight_matrices.py`: Dient zur Einstellung der Parameter der Projektionsmatrix auf den Boden: Wird verwendet, um die Gewichtsmatrix und die Maskenmatrix, die den vier überlappenden Bereichen entsprechen, zu berechnen und den Stitching-Effekt anzuzeigen.
5. `run_live_demo.py`: Wird verwendet, um die Gewichtsmatrix und die Maskenmatrix zu berechnen und den Stitching-Effekt anzuzeigen: Die endgültige Version zum Laufen auf einem echten Auto.